CompSci 275, CONSTRAINT Networks

Rina Dechter, Fall 2022

Chapter 10
Hybrid of search and inference
Time-space tradeoff

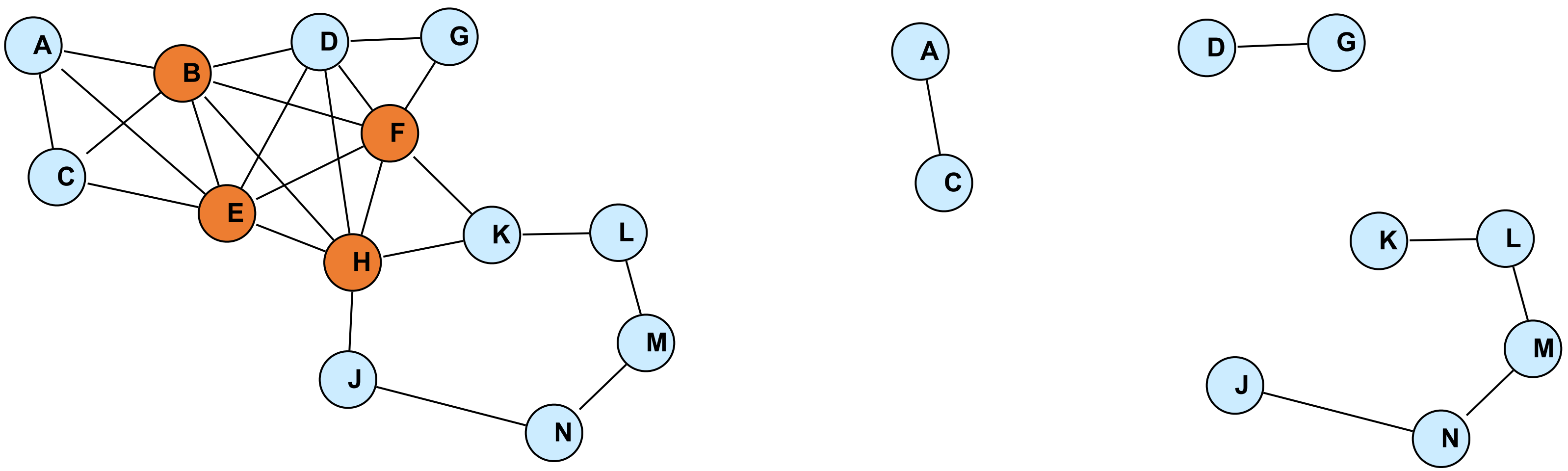
Outline

- Acyclic networks
- Join-tree clustering
- Conditioning vs tree-clustering

Transforming into a tree

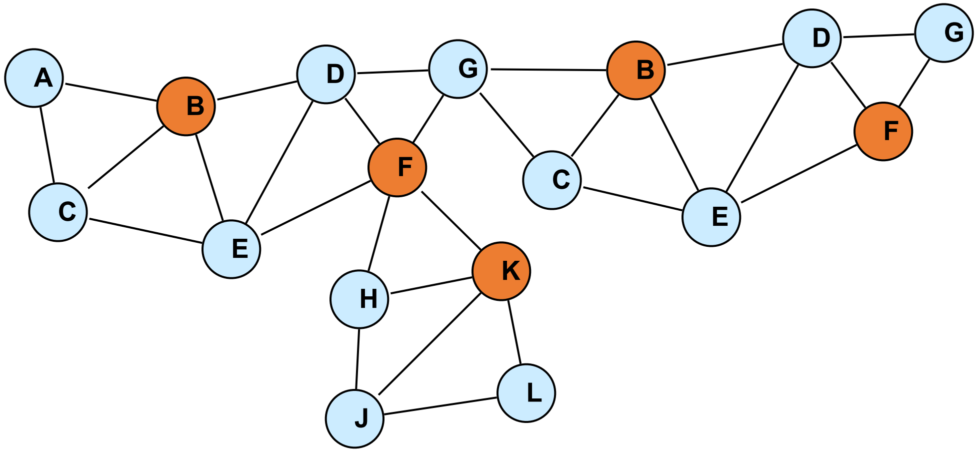
- **By Inference**
    - Time and spacde exponential in tree-width
- **By Conditioning-search**
    - Time exponential in the cycle-cutset

Treewidth equals cycle cutset



treewidth = cycle cutset = 4

Treewidth smaller than cycle cutset

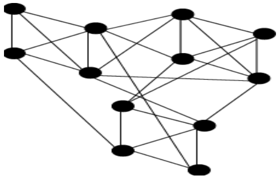


treewidth = 2

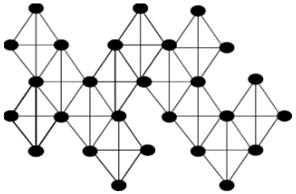
cycle cutset = 5

DR versus DPLL: Complementary Properties

Uniform random 3-CNFs
(large induced width)



(k,m)-tree 3-CNFs
(bounded induced width)



Complementary behavior of search vs variable elimination

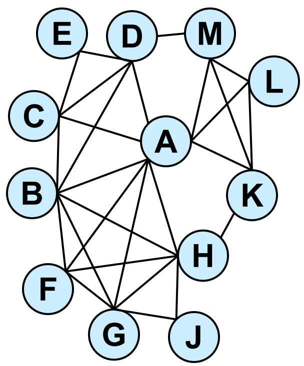
	Search	Variable Elimination
Worst-case time		
Average time	Better than worst-case	Same as worst-case
Space		
Output	One solution	Knowledge compilation

How to combine search and inference

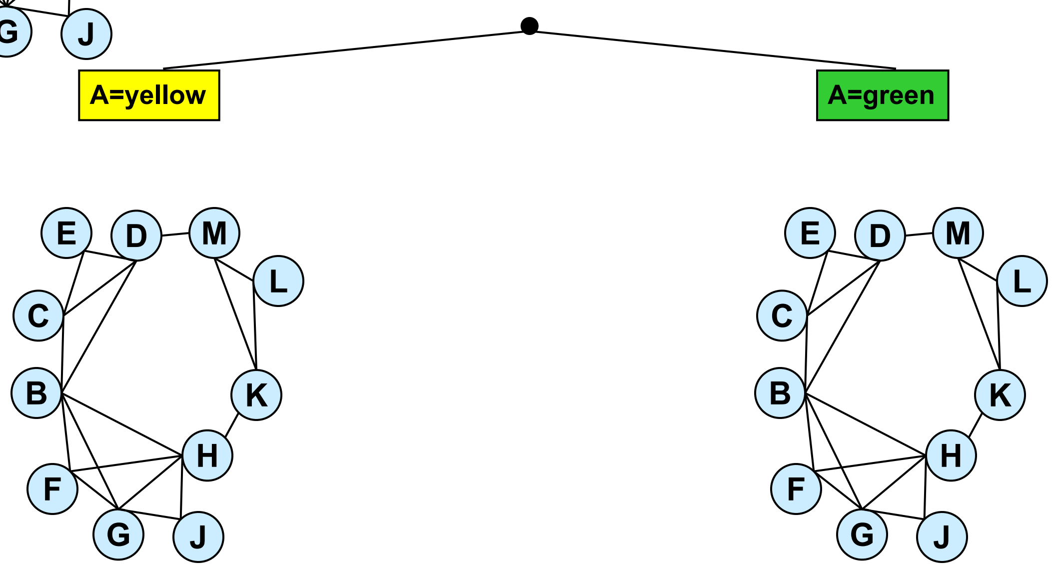
- *Pre-processing* by bounded inference and then search
- *Alternate search and inference* (do inference on a subset of variables, resulting in a smaller problem on which we can search
- Search with look-ahead (e.g arc-consistency) doing inference at every node in the search tree.
- We will illustrate 2 general architectures for hybrids.

Conditioing
Specialized cutset schemes

Graph Coloring problem

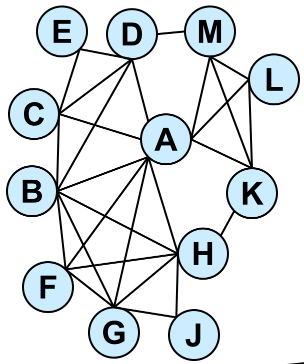


- Inference may require too much memory
- **Condition** on some of the variables

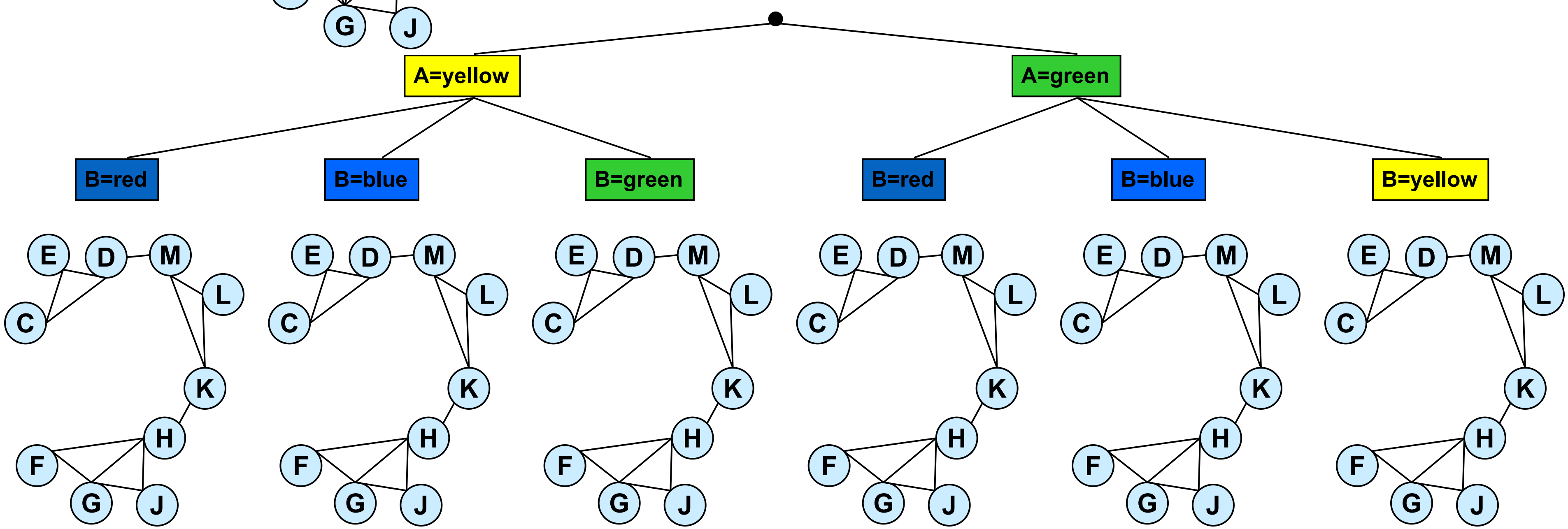


Conditioning

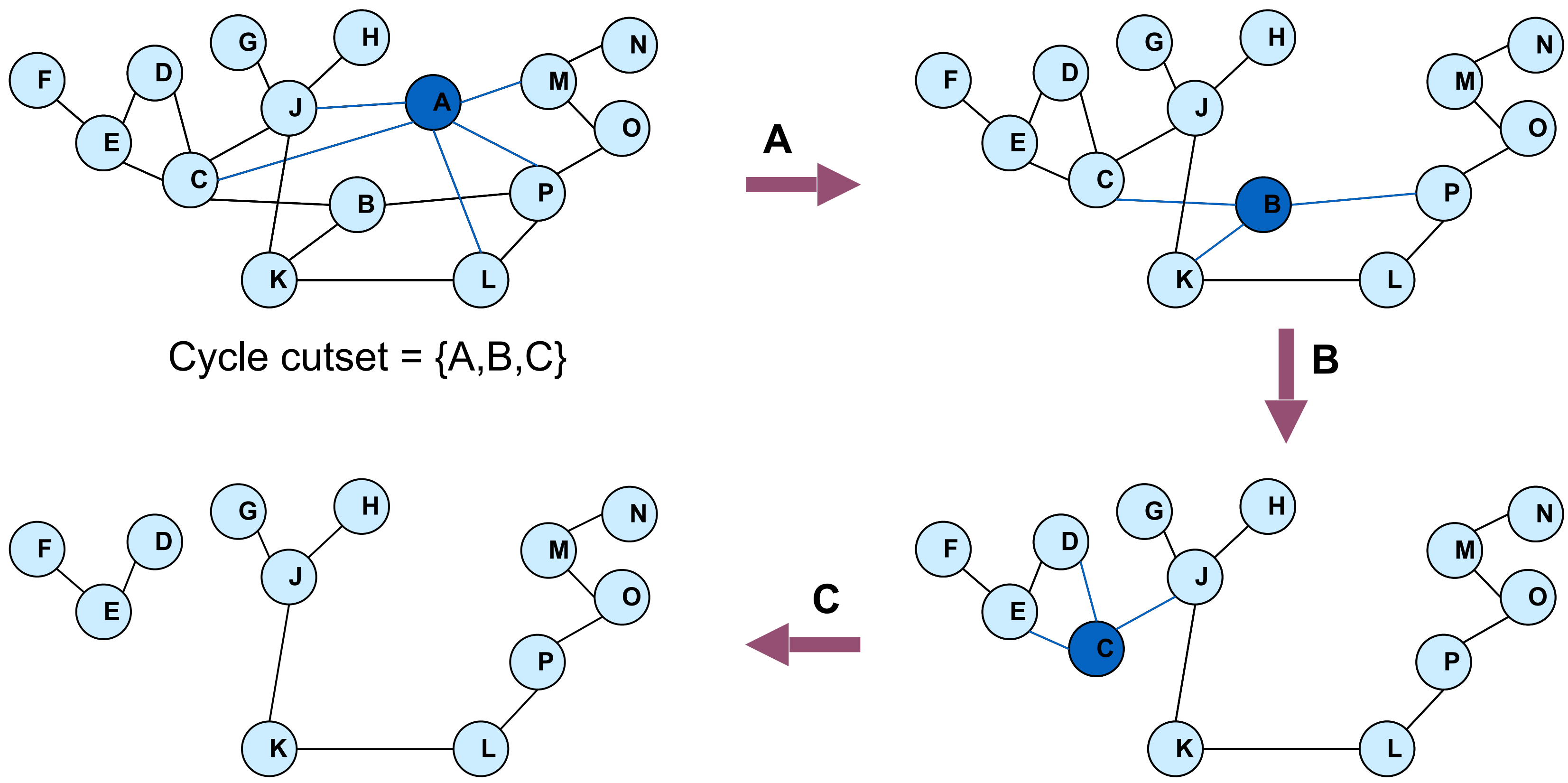
Graph Coloring problem



- Inference may require too much memory
- **Condition** on some of the variables

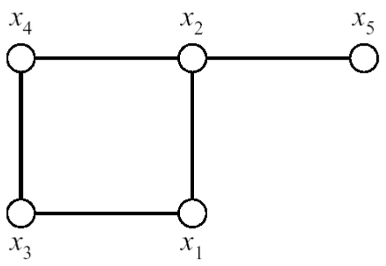


Cycle cutset

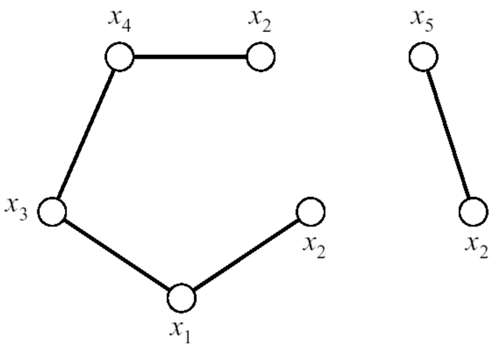


The Cycle-Cutset Effect

- A cycle-cutset is a subset of nodes in an undirected graph whose removal results in a graph with no cycles
- An instantiated variable cuts flow of information cycles
- If a cycle-cutset is instantiated the remaining problem is a tree and can be solved efficiently

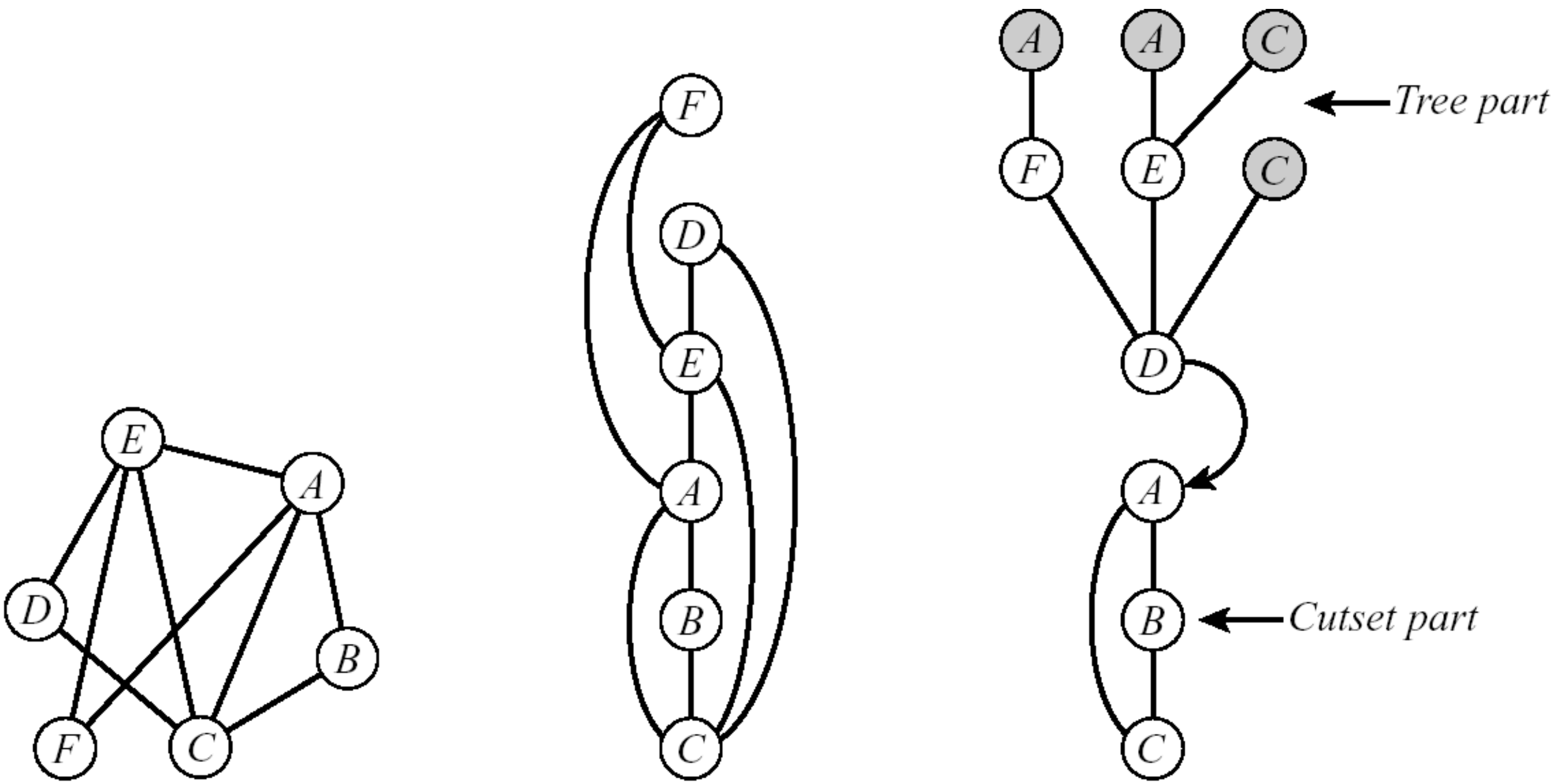


(a)



(b)

Example of the cycle-cutset scheme



(a) (b) (c)

Cycle-cutset complexity

Theorem 10.1.2 *Algorithm cycle-cutset decomposition has time complexity of* $O(nk^{c+2})$
where n is the number of variables, c is the cycle-cutset size and k is the domain size.
The space complexity of the algorithm is linear.

From cycle-cutset to b-cutset (or w-cutset)

Definition 10.2.1 (b-cutset) *Given a graph G, a subset of nodes is called a b-cutset iff when the subset is removed the resulting graph has an induced-width less than or equal to b. A minimal b-cutset of a graph has a smallest size among all b-cutsets of the graph. A cycle-cutset is a 1-cutset of a graph.*

How can we find a b-cutset? A minimal one?

Let's use a variable ordering and remove one variable after another until we get a b-cutset

Adjusted Induced-width

It is clear that finding a minimal b-cutset is a hard task, though we can define a b-cutset relative to the variable ordering. Given an ordering $d = x_1, ..., x_n$ of G, a b-cutset relative to d is obtained by processing the nodes from last to first. When node x is processed, if its induced-width is greater than b it is added to the b-cutset. Otherwise, its earlier neighbors are connected. The induced-width relative to a cutset is called *adjusted induced-width*. The adjusted induced-width relative to a b-cutset is b. Clearly a minimal b-cutset is a smallest among all b-cutsets.

Elim-Cond(b)

Algorithm elim-cond(b)
Input: A constraint network $\mathcal{R} = (X, D, C)$, $Y \subseteq X$ which is a b-cutset. d is an ordering that starts with Y such that the adjusted induced-width, relative to Y along d, is bounded by b, $Z = X - Y$.
Output: A consistent assignment, if there is one.

1. **while** $\bar{y} \leftarrow$ next partial solution of Y found by backtracking, **do**
   (a) $\bar{z} \leftarrow adaptive - consistency(\mathcal{R}_{Y=\bar{y}})$.
   (b) **if** $\bar{z}$ is not *false*, return solution$=(\bar{y}, \bar{z})$.
2. **endwhile**.
3. **return:** the problem has no solutions.

Figure 10.5: Algorithm *elim-cond(b)*

Time-space tradeoff

Theorem:

The b-cutset scheme yields space complexity $O(k^b)$and time complexity O(n k^{b+c_b}) , where c_b is the size of the b-cutset.

As b decreases, c_b increases.

The cycle-cutset decomposition is linear space and

Has time complexity of

Space-time tradeoffs

In general:

$$1 + c_1 \geq 2 + c_2 \geq \ldots b + c_b, \ldots \geq w^* + c_{w^*} = w^*$$

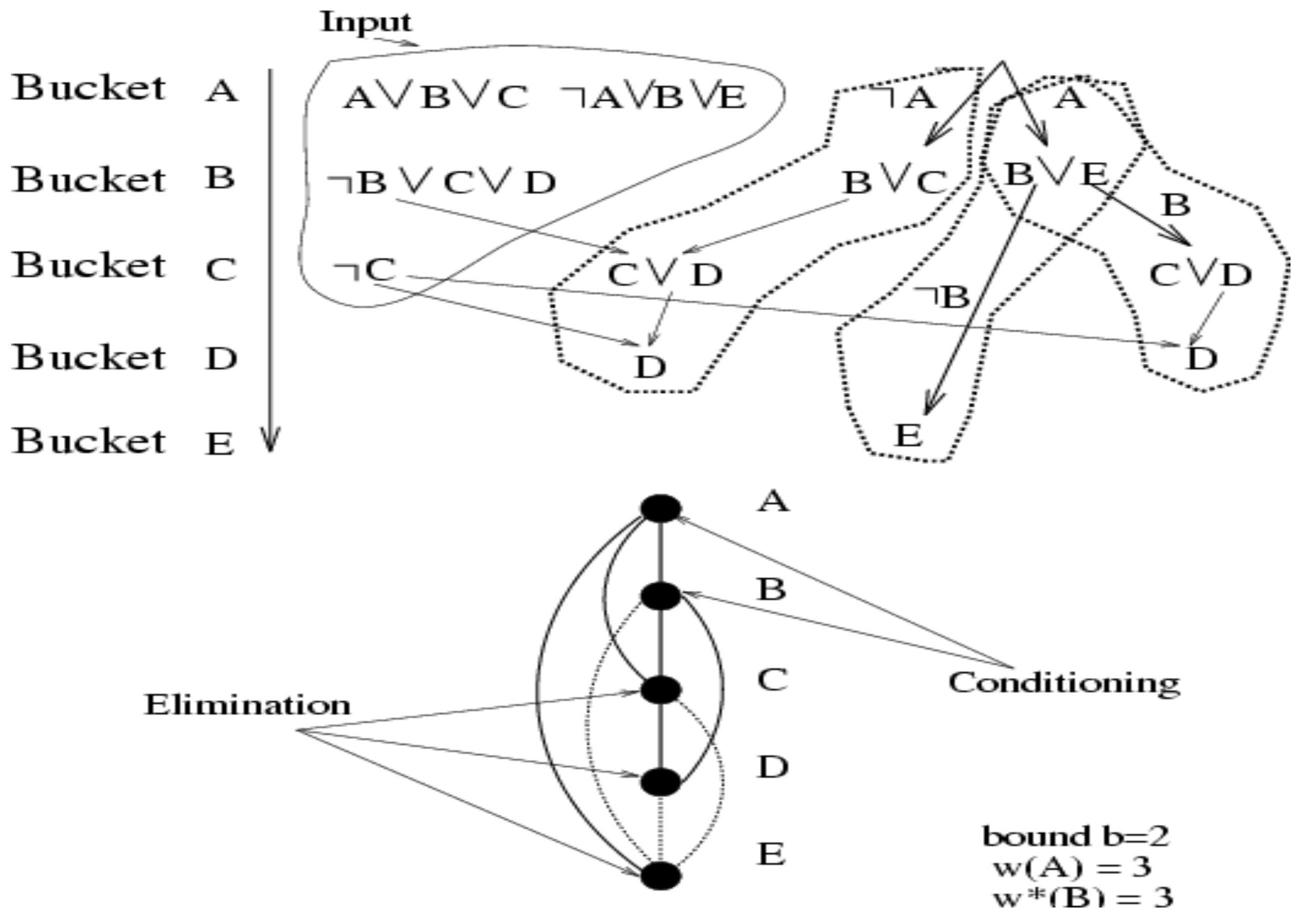
C_i is the smallest i-cutset

- Space complexity $O(k^w)$
- Time complexity $O(n\, k^{b+c_w})$, where c_w is the size of the w-cutset.
- As w decreases, c_w increases.

Hybrid algorithms for propositional theories

W-cutset Example

(~C v E)(A v B v C v D)(~A v B v E v D)(B v C v D)

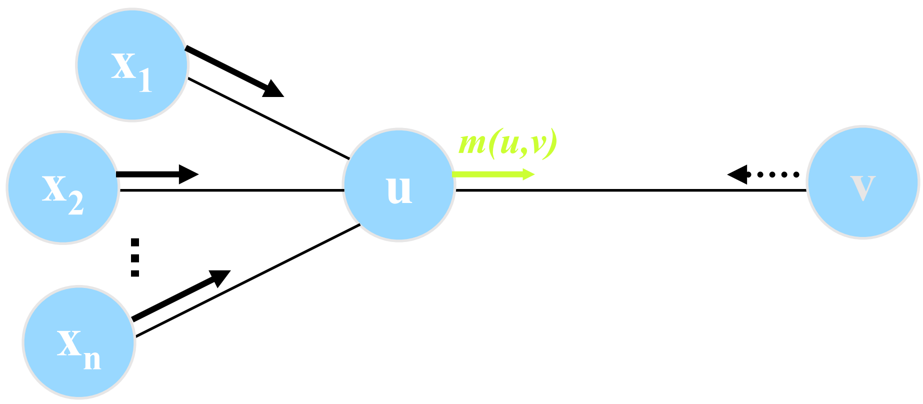


Review: Cluster Tree Elimination

- Cluster Tree Elimination (CTE) works by passing messages along a tree-decomposition

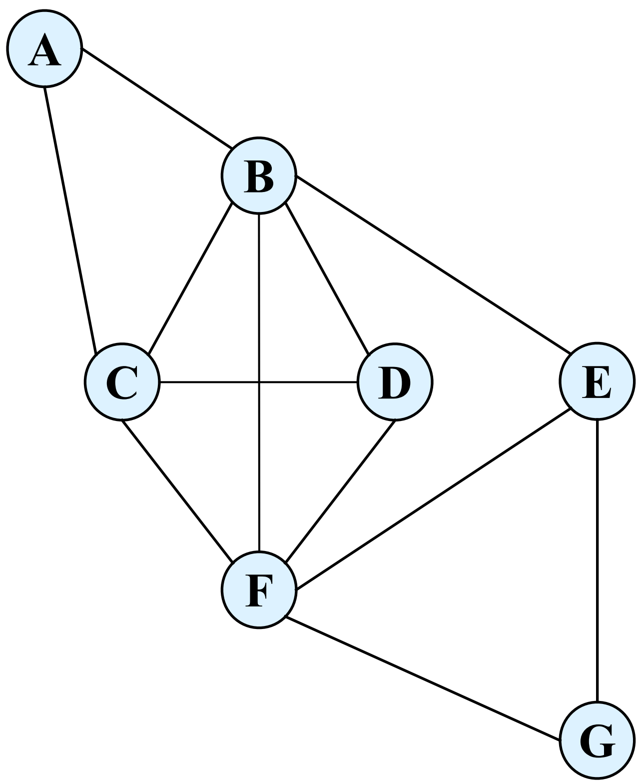
- Basic idea:
  - Each node sends one message to each of its neighbors
  - Node *u* sends a message to its neighbor *v* only when *u* received messages from all its other neighbors

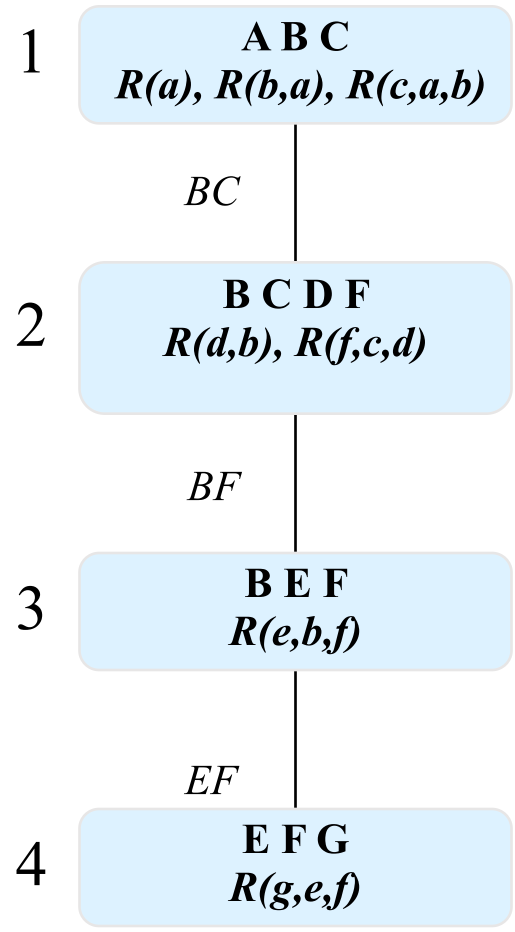
Constraint propagation



Join-Tree Decomposition

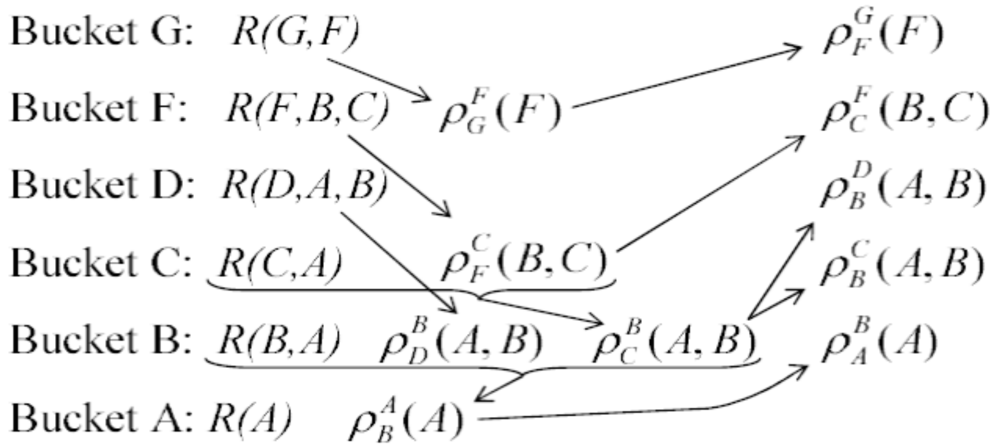
(Dechter and Pearl 1989)

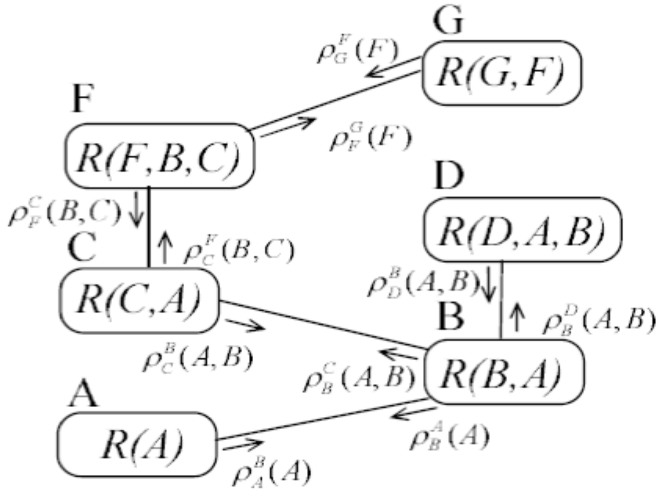


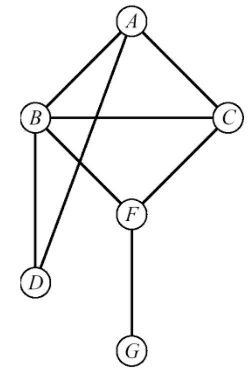


- Each function in a cluster
- Satisfy running intersection property
- Tree-width: number of variables in a cluster-1
- Equals induced-width

The bottom up messages



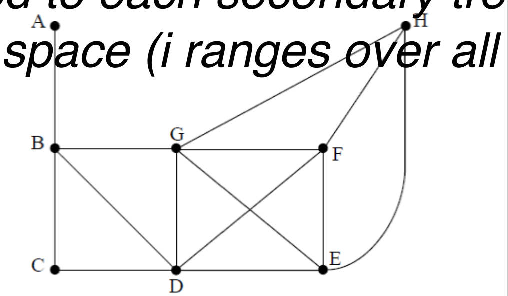




(a)

Super-clustering

Theorem 10.3.1 *Given a tree-decomposition T over n variables, separator sizes* $s_0; s_1 \ldots, s_t$
and secondary tree-decompositions having a corresponding maximal number of nodes in
any cluster, $r0, r1, \ldots, rt$. *The complexity of CTE when applied to each secondary tree-decompositions T_i is O(n exp(r_i)) time, and O(n exp(s_i)) space (i ranges over all the*
secondary tree-decomposition).



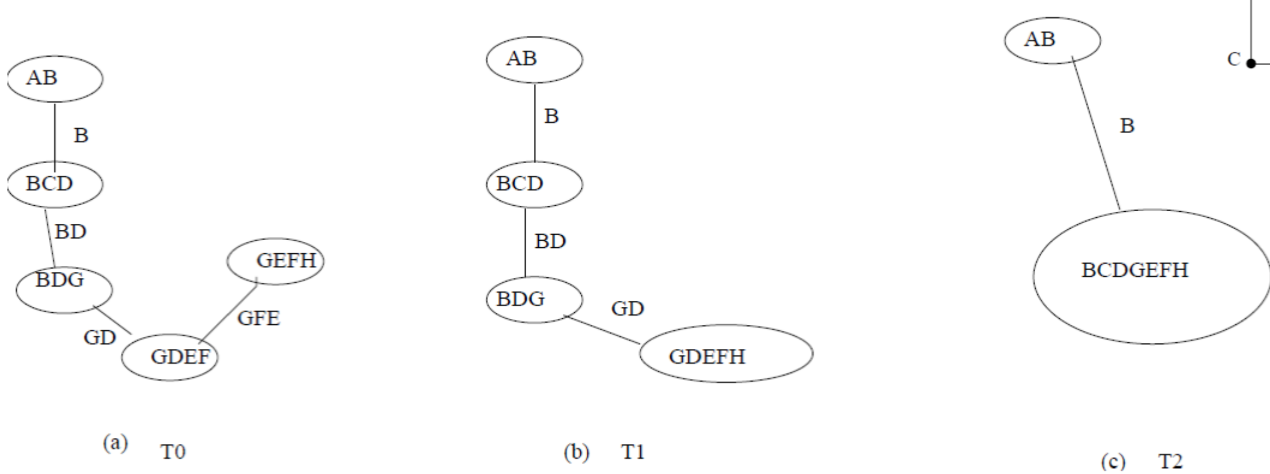
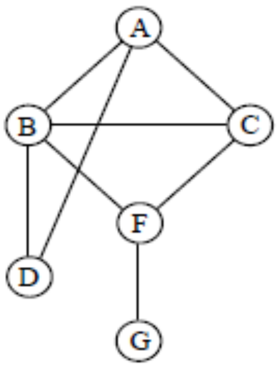


Figure 10.12: A tree-decomposition with separators equal to (a) 3, (b) 2, and (c) 1

Super-buckets



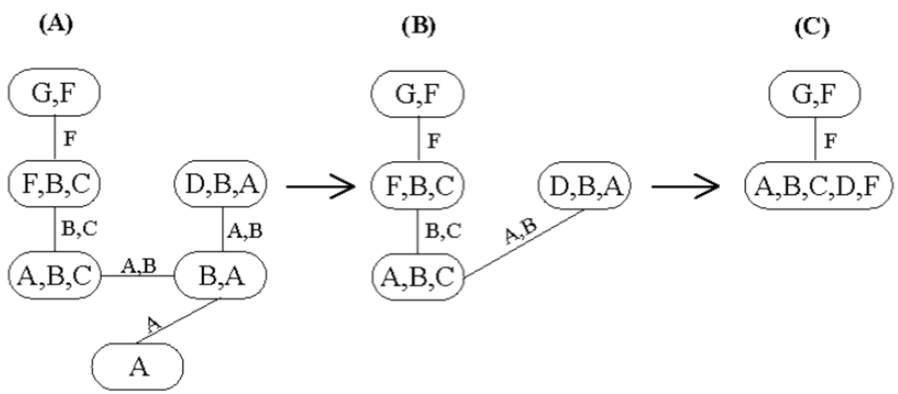
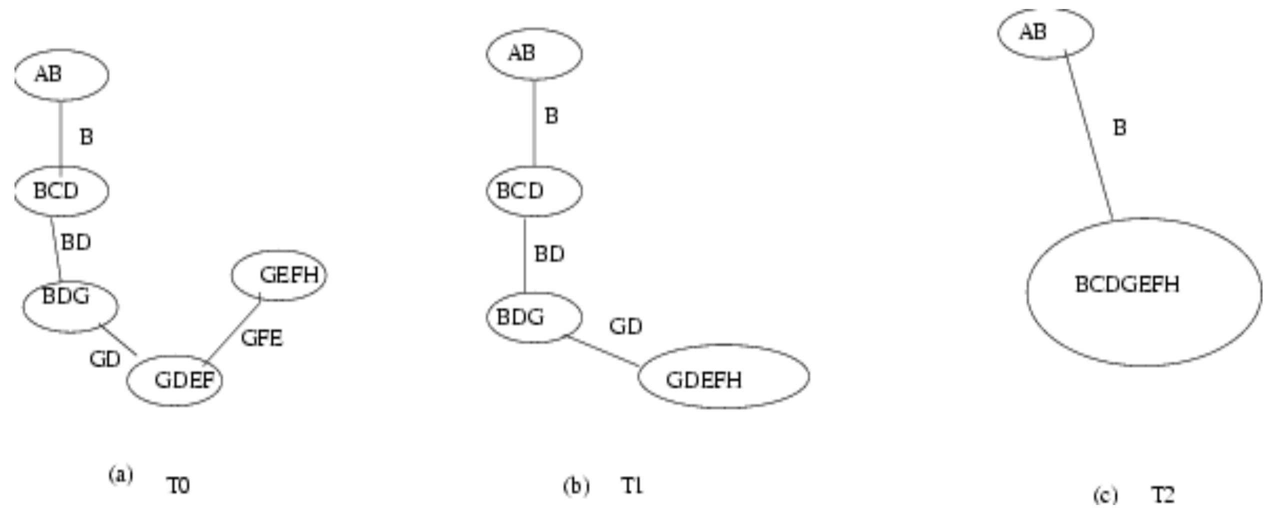


Figure 10.13: From a bucket-tree (left) to join-tree (middle) to a super-bucket-tree (right)

The idea of super-buckets/clusters

Larger super-buckets (cliques) =>more time but less space



Complexity:
1. Time: exponential in clique (super-bucket) size
2. Space: exponential in separator size

Algorithm CTE(b)" generate a tree-decomposition with seperator size bound by b. Then, apply algorithm CTE.

Hybrid of Hybrids.

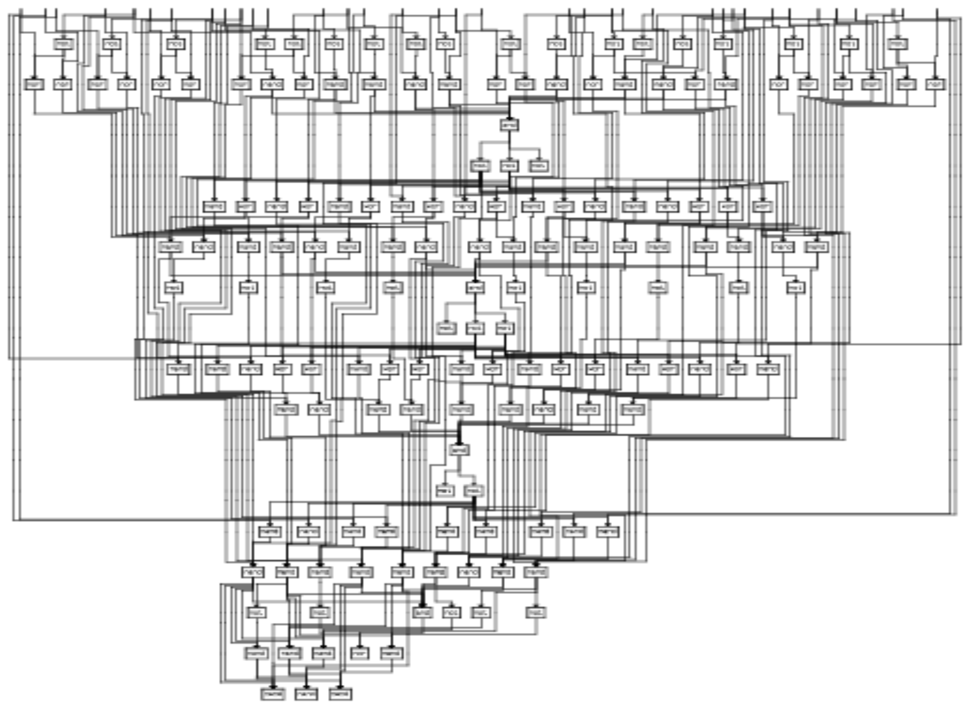
Algorithm hybrid(*b*1*; b*2): Apply elim-cond(b2) inside each cluster of a b-tree-decomposition.
Space complexity exp(b1) time complexity exp(b2), b2 <= b1.

Hybrid(b_1,1): apply cycle-cutset in each cluster (see circuit examples next)

Hybrid(1,1): apply cycle-cutset for each non-separable component.

Application: circuit diagnosis

Problem: Given a circuit and its unexpected output, identify faulty components. The problem can be modeled as a constraint optimization problem and solved by bucket elimination.



Circuit	c17	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c6288	c7552
#nodes	11	196	243	443	587	913	1426	1719	2485	2448	3719
#edges	18	660	692	1140	1660	2507	3226	4787	7320	7184	9572

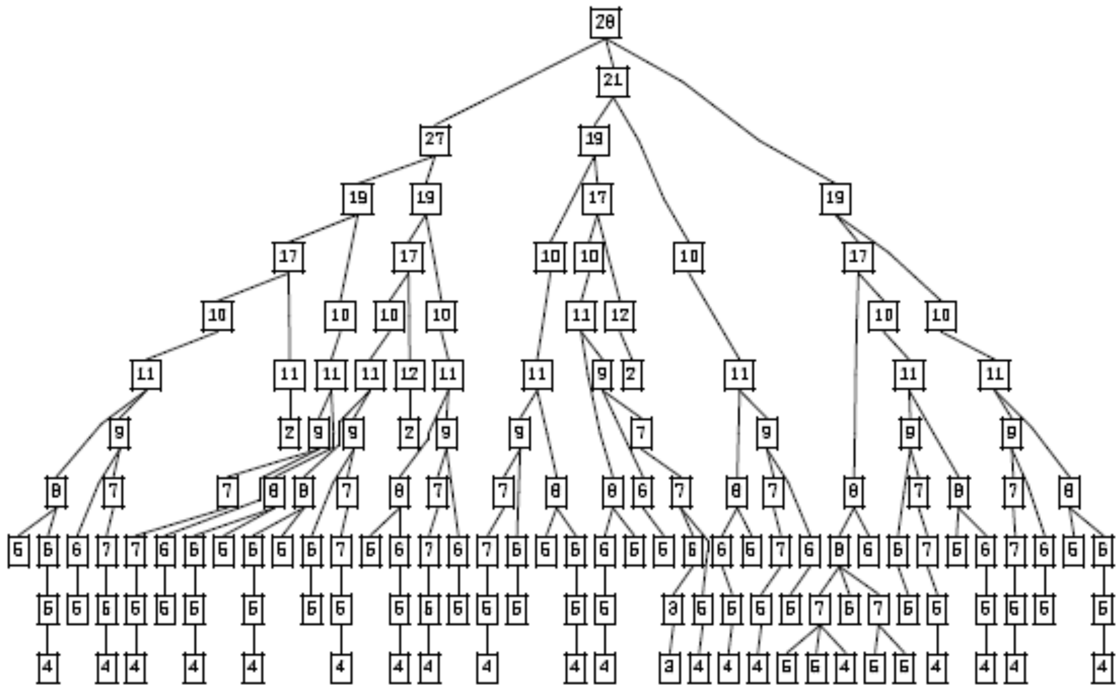


Figure 10.17: Primary join tree (157 cliques) for circuit c432 (196 variables); the maximum separator size is 23.

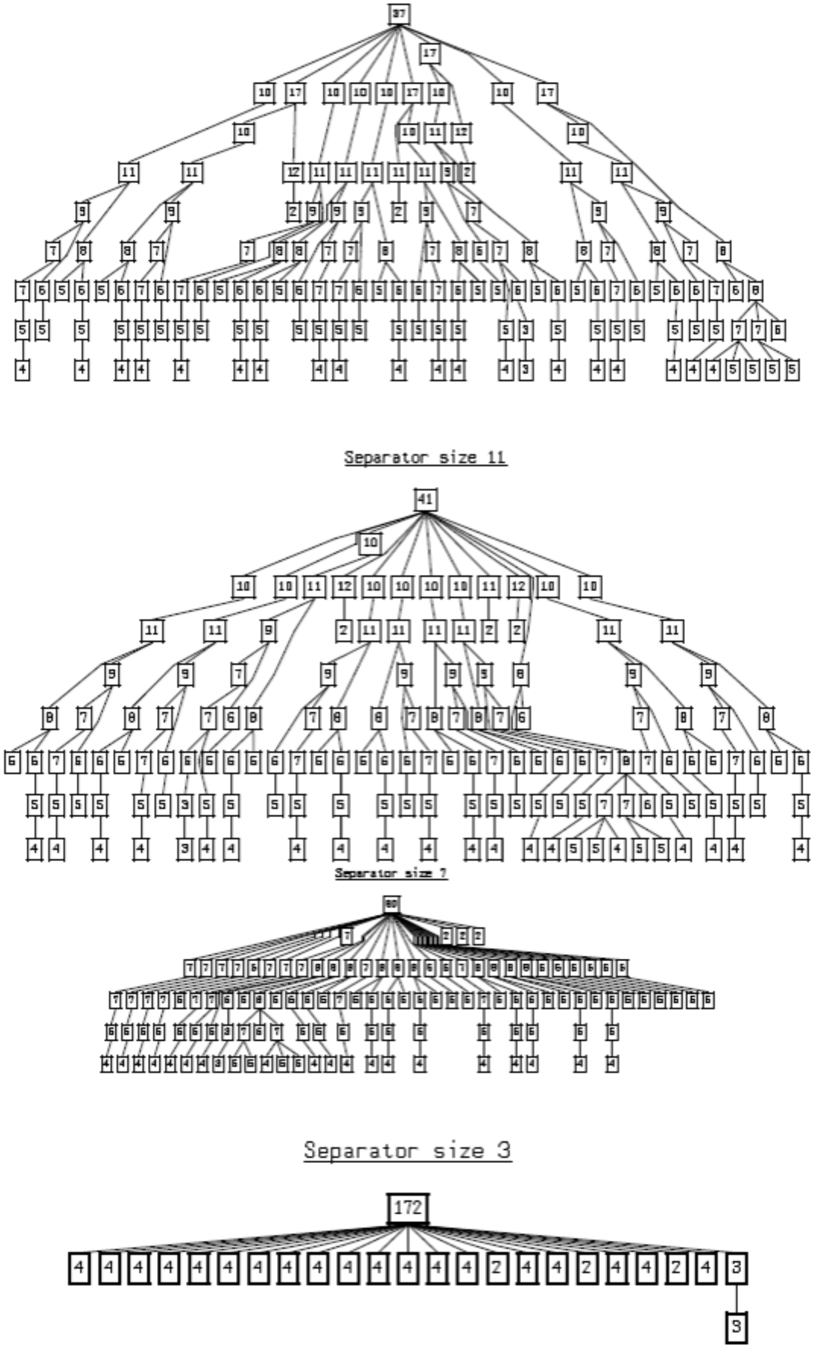


Figure 10.20: Secondary trees for c432 with separator sizes 16 and 11, 7 and 3.

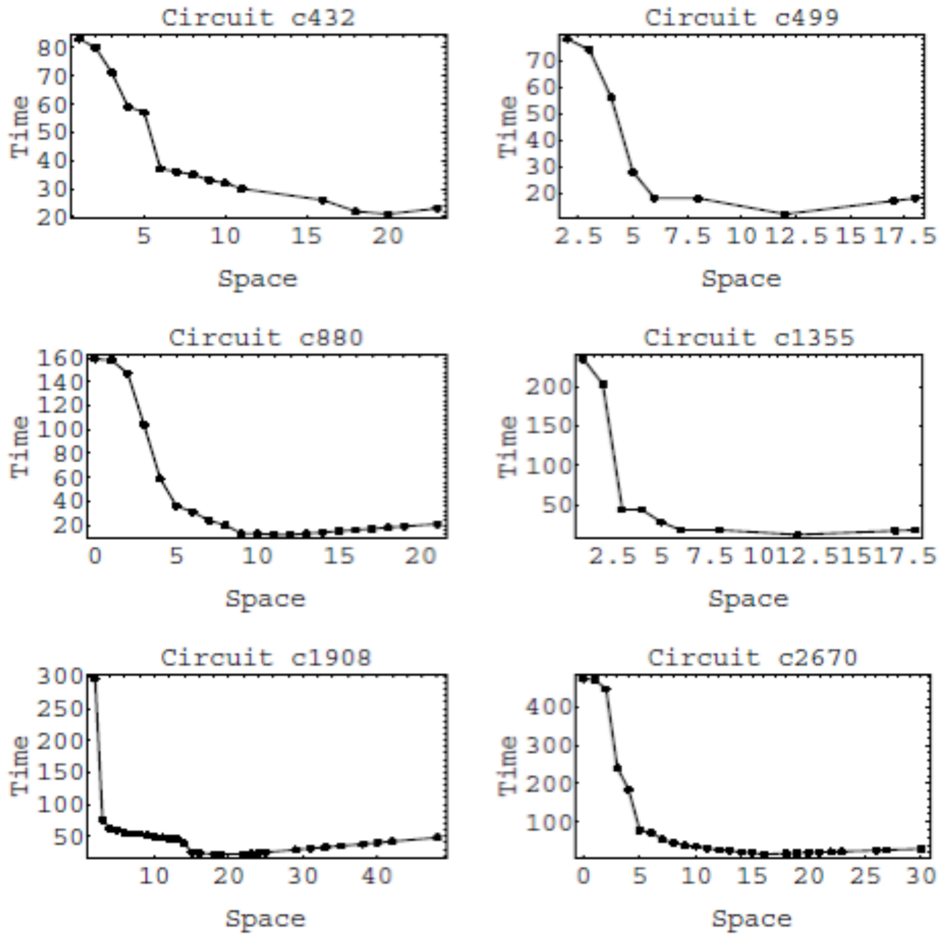


Figure 10.19: Time/Space tradeoff for c432 (196 variables), c499 (243 variables), c880 (443 variables), c1355 (587 variables), c1908 (913 variables) and c2670 (1426 variables). Time is measured by the maximum of the separator size and the cutset size and space by the maximum separator size.

CompSci 275, Constraint Networks

Rina Dechter, Fall 2022

Constraint Optimization and counting
Chapter 13

Outline

- **Introduction**
  - Optimization tasks for graphical models
  - Solving optimization problems with inference and search
- **Inference**
  - Bucket elimination, dynamic programming
  - Mini-bucket elimination
- **Search**
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - AND/OR search spaces

Outline

- **Introduction**
  - Optimization tasks for graphical models
  - Solving optimization problems with inference and search
- **Inference**
  - Bucket elimination, dynamic programming
  - Mini-bucket elimination
- **Search**
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - AND/OR search spaces

Constraint optimization problems
for graphical models

f(A,B,D) has scope {A,B,D}

A	B	D	Cost
1	2	3	3
1	3	2	2
2	1	3	0
2	3	1	0
3	1	2	5
3	2	1	0

Constraint Optimization Problems
for Graphical Models

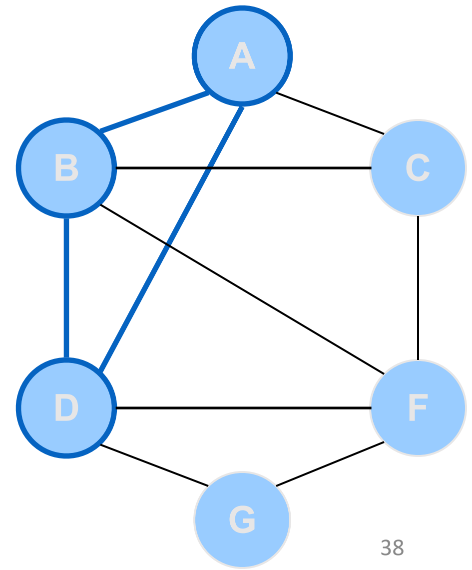
f(A,B,D) has scope {A,B,D}

A	B	D	Cost
1	2	3	3
1	3	2	2
2	1	3	0
2	3	1	0
3	1	2	5
3	2	1	0

Primal graph =
Variables --> nodes
Functions, Constraints -→ arcs

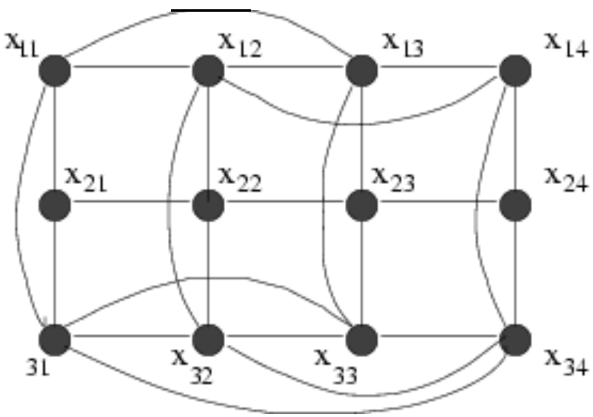
F(A,B,C,D,F,G)= f1(A,B,D)+f2(D,F,G)+f3(B,C,F)

f1(A,B,D)
f2(D,F,G)
f3(B,C,F)



Example: constrained optimization

Example: power plant scheduling



Unit #	Min Up Time	Min Down Time
1	3	2
2	2	1
3	4	1

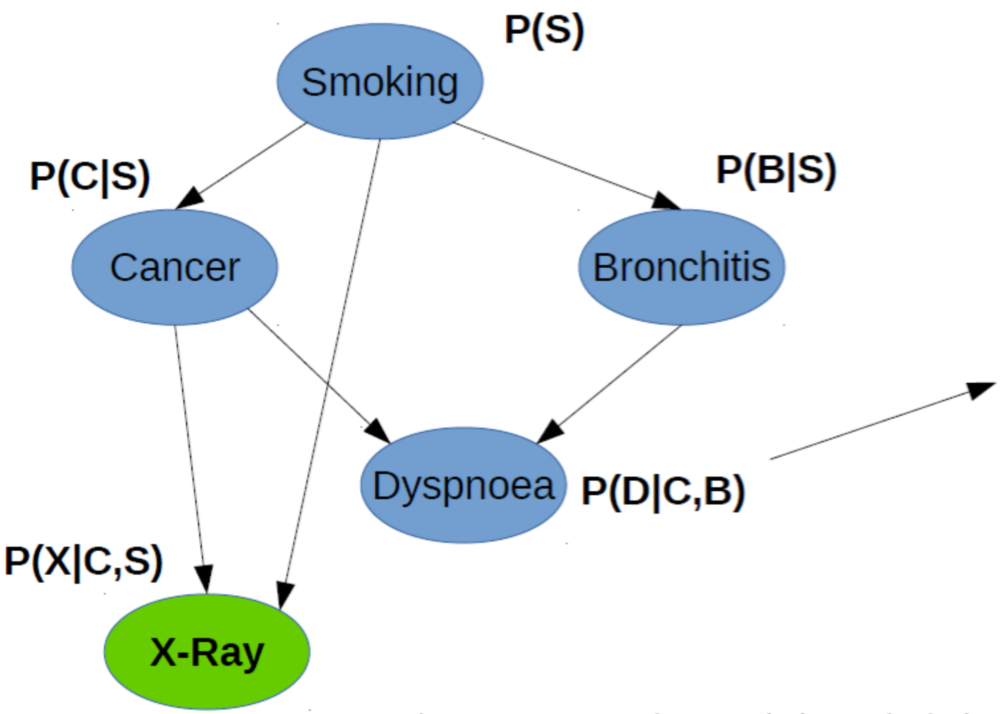
Example: combinatorial auction

Given a set of elements A= $\{a_1,\dots, a_n\}$ and given a set of bids, B= $\{b_1,\dots, b_l\}$, each is a subset of A, each having cost r_i, select a subset of bids B’ with maximal total cost where no two bids share an item.

$$max_{B'} Max \sum_{b_i \ in \ B'} r\_i$$

Consider a problem instance given by the following bids: $b_1 = \{1, 2, 3, 4, \}$, $b_2 = \{2, 3, 6\}$, $b_3 = \{1, 5, 4\}$, $b_4 = \{2, 8\}$, $b_5 = \{5, 6\}$ and the costs $r_1 = 8$, $r_2 = 6$, $r_3 = 5$, $r_4 = 2$, $r_5 = 2$. In this case the variables are b_1, b_2, b_3, b_4, b_5, their domains are $\{0, 1\}$ and the constraints are $R_{12}, R_{13}, R_{14}, R_{24}, R_{25}, R_{35}$. The cost network for this problem formulation is identical to its constraint network, since all the cost components are unary. The reader can verify that an optimal solution is given by $b_1 = 0, b_2 = 1, b_3 = 1, b_4 = 0, b_5 = 0$. Namely, selecting bids b_2 and b_3 is an optimal choice with total cost of 11. □

Probabilistic Networks



P(D|C,B)

C	B	D=1	D=0
0	0	0.1	0.9
0	1	0.7	0.3
1	0	0.8	0.2
1	1	0.9	0.1

$$P(S, C, B, X, D) = P(S) \cdot P(C|S) \cdot P(B|S) \cdot P(X|C, S) \cdot P(D|C, B)$$

MPE: Find a maximum probability assignment, given evidence

= Find $\arg\max P(S) \cdot P(C|S) \cdot P(B|S) \cdot P(X|C, S) \cdot P(D|C, B)$

Graphical models

- A graphical model **(X,D,F)**:
  - **X** = $\{X_1,\ldots X_n\}$ variables
  - **D** = $\{D_1, \ldots D_n\}$ domains
  - **F** = $\{f_1,\ldots,f_m\}$ functions

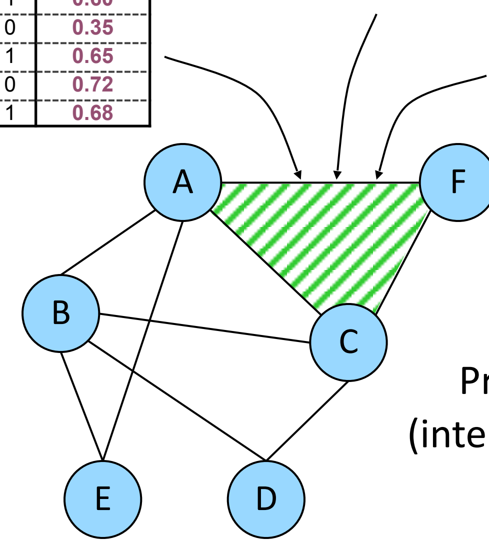
- Operators:
  - combination
  - elimination (projection)

- Tasks:
  - **Belief updating**: $\Sigma_{X-y} \prod_j P_i$
  - **MPE**: $\max_X \prod_j P_j$
  - **CSP**: $\prod_X \times_j C_j$
  - **Max-CSP**: $\min_X \Sigma_j f_j$

A	C	F	P(F|A,C)
0	0	0	0.14
0	0	1	0.96
0	1	0	0.40
0	1	1	0.60
1	0	0	0.35
1	0	1	0.65
1	1	0	0.72
1	1	1	0.68

Relation

A	C	F
red	green	blue
blue	red	red
blue	blue	green
green	red	blue



Primal graph
(interaction graph)

- All these tasks are NP-hard
  - exploit problem structure
  - identify special cases
  - approximate

Combination of cost functions

A	B	f(A,B)
b	b	6
b	g	0
g	b	0
g	g	6

+

B	C	f(B,C)
b	b	6
b	g	0
g	b	0
g	g	6

A	B	C	f(A,B,C)
b	b	b	12
b	b	g	6
b	g	b	0
b	g	g	6
g	b	b	6
g	b	g	0
g	g	b	6
g	g	g	12

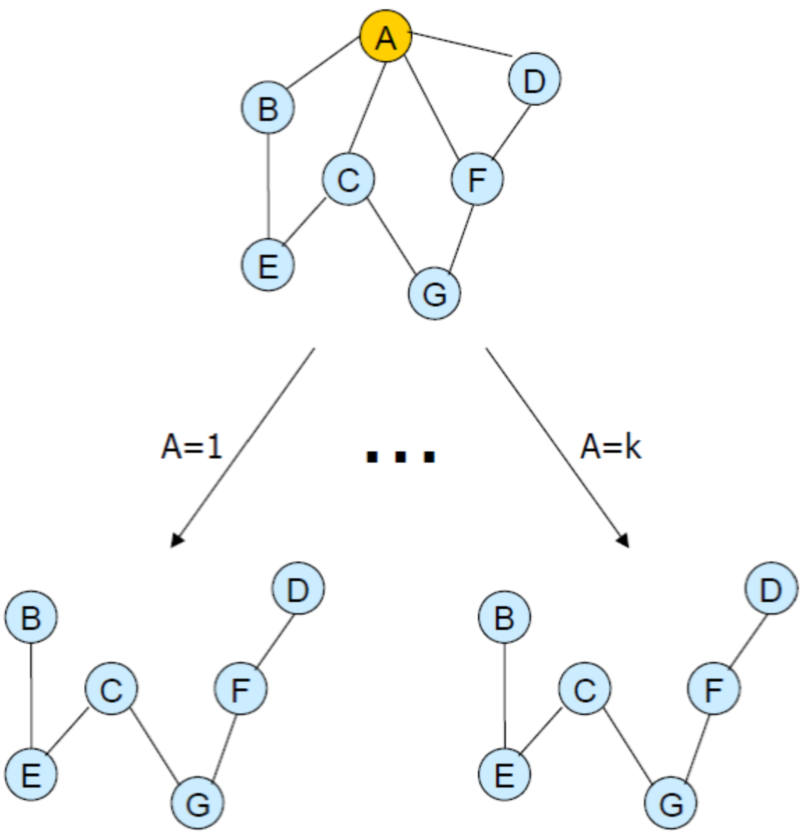
= 0 + 6

Outline

- **Introduction**
  - Optimization tasks for graphical models
  - Solving by inference and search
- **Inference**
  - **Bucket elimination, dynamic programming, tree-clustering, bucket-elimination**
  - Mini-bucket elimination, belief propagation
- **Search**
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - AND/OR search spaces
- **Hybrids of search and inference**
  - Cutset decomposition
  - Super-bucket scheme

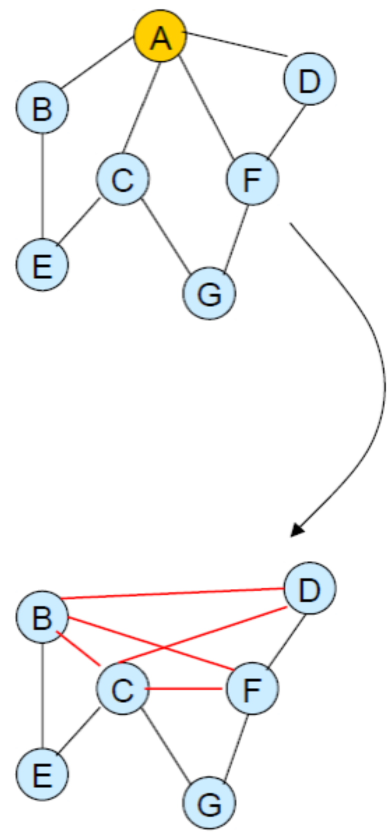
Conditioning vs. Elimination

Conditioning (search)



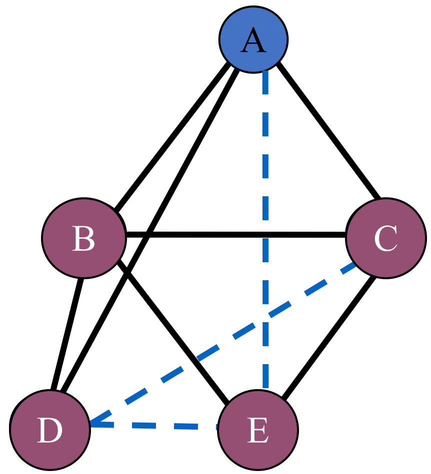
k "sparser" problems

Elimination (inference)



1 "denser" problem

Computing the optimal cost solution



Constraint graph

OPT = $f(a,b)+f(a,c)+f(a,d)+f(b,c)+f(b,d)+f(b,e)+f(c,e)$

Combination

$f(a,d) + \quad f(a,c)+f(c,e) + \quad f(a,b)+f(b,c)+f(b,d)+f(b,e)$

Variable Elimination

Finding

Algorithm **elim-opt** (Dechter, 1996)
Non-serial Dynamic Programming (Bertele and Briochi, 1973)

Elimination operator

bucket B: f(a,b) f(b,c) f(b,d) f(b,e)

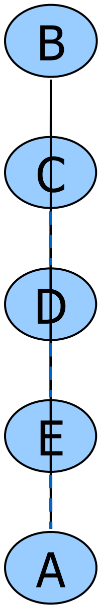
bucket C: f(c,a) f(c,e)

bucket D: f(a,d)

bucket E: e=0

bucket A:

OPT



Generating the optimal assignment

B: f(a,b) f(b,c) f(b,d) f(b,e)

C: f(c,a) f(c,e)

D: f(a,d)

E: e=0

A:

Complexity

Algorithm **elim-opt** (Dechter, 1996)
Non-serial Dynamic Programming (Bertele and Briochi, 1973)

Elimination operator

bucket B: f(a,b) f(b,c) f(b,d) f(b,e)

bucket C: f(c,a) f(c,e)

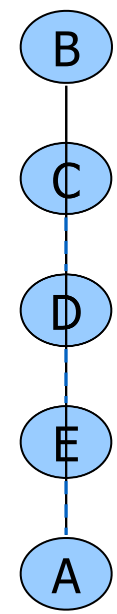
bucket D: f(a,d)

bucket E: e=0

bucket A:

OPT

exp(w*=4)
"induced width"
(max clique size)

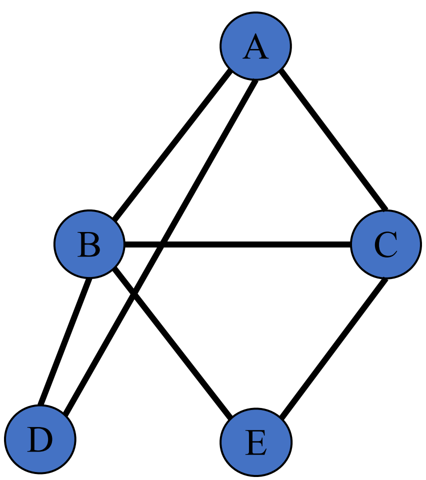


Induced width

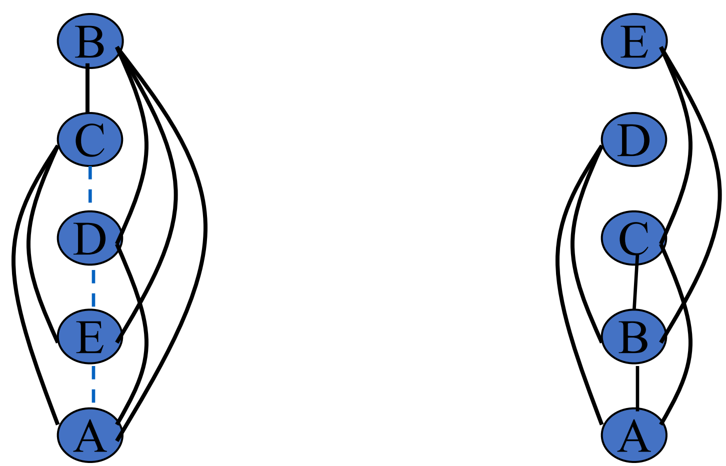
Bucket-elimination is **time** and **space**

r = number of functions

The effect of the ordering:



constraint graph



Finding smallest induced-width is hard

Using a different ordering

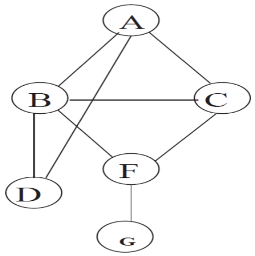


Figure 13.1: The cost graph of the cost function: $C(a,b,c,d,f,g) = F_0(a) + F_1(a,b) + F_2(a,c) + F_3(b,c,f) + F_4(a,b,d) + F_5(f,g)$

Bucket G: $F_5(g,f)$

Bucket D: $F_4(d,b,a)$

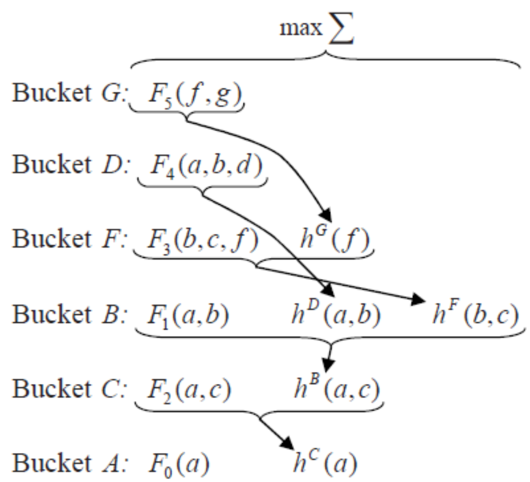
Bucket F: $F_3(f,b,c)$

Bucket B: $F_1(b,a)$

Bucket C: $F_2(c,a)$

Bucket A: $F_0(a)$

(a)



(b)

Figure 13.4: Bucket elimination along ordering $d_1 = A, C, B, F, D, G$.

Induced-width (again...)

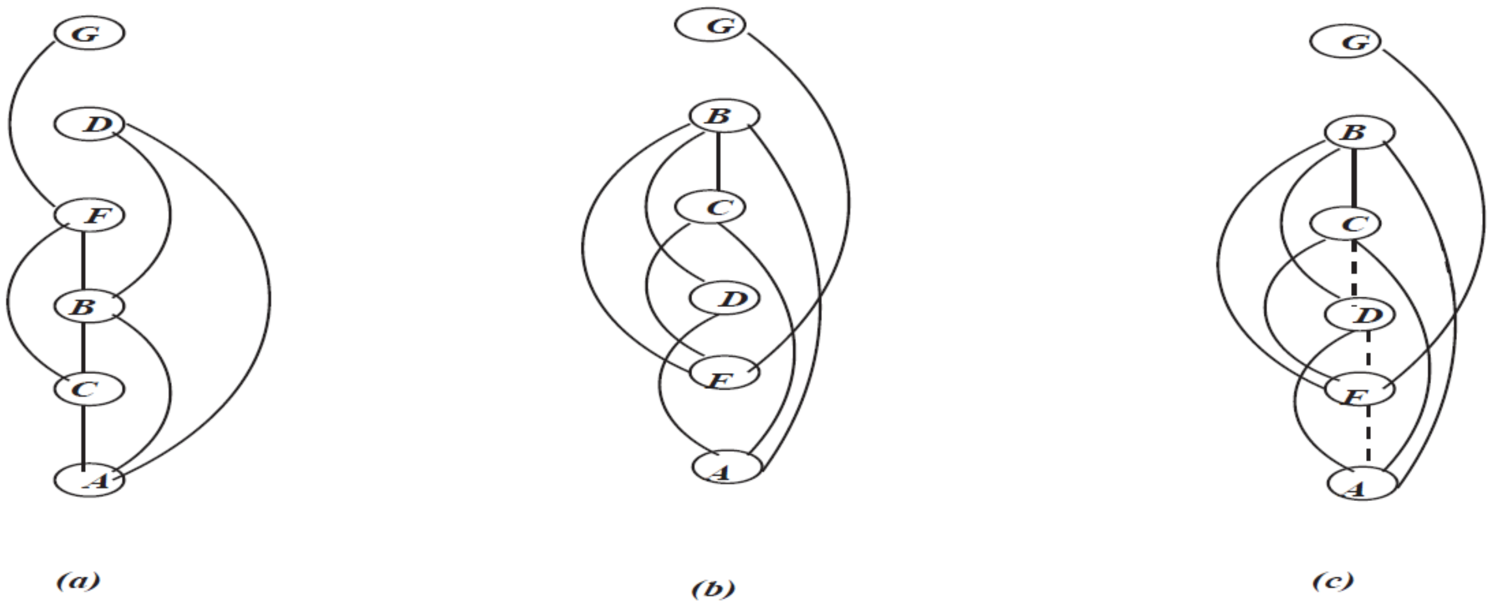


Figure 13.7: Two orderings of the cost graph of our example problem

Min sum
Is more popular

Elim-opt: BE for optimization

Algorithm elim-opt
Input: A cost network $\mathcal{C} = (X, D, C)$, $C = \{F_1, ..., F_l\}$; ordering d
Output: The maximal cost assignment to $\sum_j F_j$.
1. **Initialize:** Partition the cost components into ordered buckets.
2. **Process buckets** from $p \leftarrow n$ downto 1
For costs $h_1, h_2, ..., h_j$ defined over scopes $Q_1, ..., Q_j$ in $bucket_p$, do:

- **If** (observed variable) $x_p = a_p$, assign $x_p = a_p$ to each h_i and put in appropriate buckets. Terminate if value is inconsistent.
- **Else,** (sum and maximize)
  $A \leftarrow \cup_i Q_i - \{x_p\}$
  $h^p = max_{x_p} \sum_{i=1}^{j} h_i$.
  Place h^p in the latest lower bucket mentioning a variable in A.

3. **Forward:** From $i = 1$ to n, given $\vec{a}_{i-1}$, assign x_i a value a_i that maximizes the sum values of functions in its bucket.

Figure 13.5: Dynamic programming as elim-opt

Example: combinatorial auction

Given a set of elements A= $\{a_1,\ldots, a_n\}$ and given a set of bids, B= $\{b_1,\ldots, b_l\}$, each is a subset of A, each having cost r_i, select a subset of bids B' with maximal total cost where no two bids share an item.

$$max_{B'} Max \sum_{b_i \; in \; B'} r\_i$$

Consider a problem instance given by the following bids: $b_1 = \{1, 2, 3, 4, \}$, $b_2 = \{2, 3, 6\}$, $b_3 = \{1, 5, 4\}$, $b_4 = \{2, 8\}$, $b_5 = \{5, 6\}$ and the costs $r_1 = 8$, $r_2 = 6$, $r_3 = 5$, $r_4 = 2$, $r_5 = 2$. In this case the variables are b_1, b_2, b_3, b_4, b_5, their domains are $\{0, 1\}$ and the constraints are $R_{12}, R_{13}, R_{14}, R_{24}, R_{25}, R_{35}$. The cost network for this problem formulation is identical to its constraint network, since all the cost components are unary. The reader can verify that an optimal solution is given by $b_1 = 0, b_2 = 1, b_3 = 1, b_4 = 0, b_5 = 0$. Namely, selecting bids b_2 and b_3 is an optimal choice with total cost of 11. $\square$

Elim-opt for auction problem

processing bucket b_4. In this bucket we compute $h^4(b_1, b_2) = \max_{\{(b_4|(b_1,b_2,b_4)\in R_{41}\bowtie R_{42}\}} r(b_4)$, yielding:

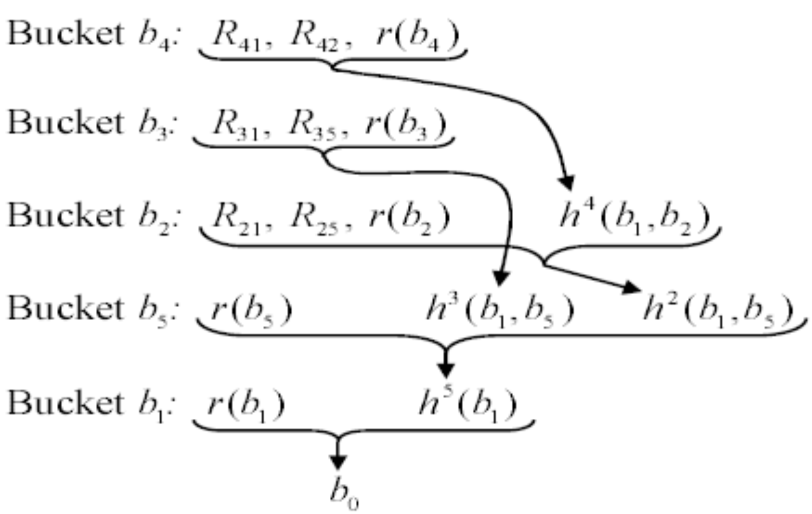
$$h^4(b_1, b_2) = \begin{cases} 0 & \text{if } b_1 = 1, \ or \ b_2 = 1 \\ 2 & \text{if } b_1 = 0, b_2 = 0 \end{cases}$$

Processing bucket b_3 we compute $h^3(b_1, b_5) = \max_{\{(b_3|(b_1,b_3,b_5)\in R_{31}\bowtie R_{35}\}} r(b_3)$, yielding:

$$h^3(b_1, b_5) = \begin{cases} 0 & \text{if } b_1 = 1, \ or \ b_5 = 1 \\ 5 & \text{if } b_1 = 0, b_5 = 0 \end{cases}$$

Processing bucket b_2, which now includes a new function, gives us:
$h^2(b_1, b_5) = \max_{\{(b_2|(b_1,b_2,b_5)\in R_{21}\bowtie R_{25}\}}(r(b_2) + h^4(b_1, b_2))$, yielding:

$$h^2(b_1, b_5) = \begin{cases} 0 & \text{if } b_1 = 1, b_5 = 1 \\ 0 & \text{if } b_1 = 1, b_5 = 0 \\ 2 & \text{if } b_1 = 0, b_5 = 1 \\ 6 & \text{if } b_1 = 0, b_5 = 0 \end{cases}$$



(a)

*b*1 = {1, 2, 3, 4}; *b*2 ={2, 3, 6}; *b*3 = {1, 5, 4} *b*4 = {2, 8} *b*5 = {5, 6}

costs *r*1 = 8; *r*2 = 6; *r*3 = 5; *r*4 =2; *r*5 = 2

Bucket-elimination for combinatorial auction

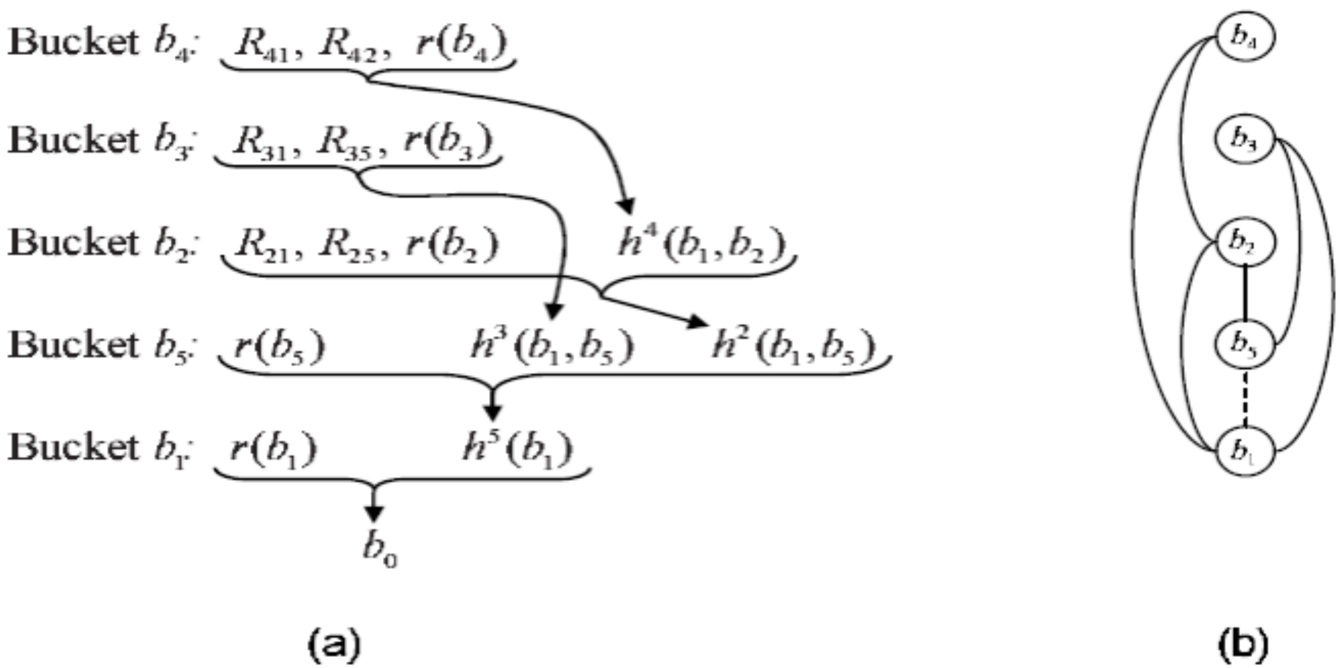


Figure 13.8: Schematic execution of elim-opt on the auction problem

Treating constraints as constraints

Algorithm elim-opt-cons
Input: A cost network $\mathcal{C} = (X, D, C_h, C_s)$, $C_h = \{R_{S_1}, ..., R_{S_m}\}$; $C_s = \{F_{Q_1}, ..., F_{Q_l}\}$. ordering d;
Output: A consistent solution that maximizes $\sum_{F_i \in C_s} F_i$.
1. **Initialize:** Partition the C_s and C_h into buckets using the usual rule.

2. **Process buckets** from $p \leftarrow n$ downto 1,
For costs $h_1, h_2, ..., h_j$ defined over scopes $Q_1, ..., Q_j$, for hard constraint relations $R_1, R_2, ..., R_t$ defined over scopes $S_1, ..., S_t$ in $bucket_p$, do:

- **If** (observed variable) $x_p = a_p$, assign $x_p = a_p$ to each h_i and each R_i and put in appropriate buckets.
- **Else,** (sum and maximize, join and project)
  1. Let $U_p = \cup_i S_i - \{x_p\}$, $V_p = \cup_i Q_i - \{x_p\}$, $W_p = U_p \cup V_p$
  2. $R^p = \pi_{U_p}(\Join_{i=1}^{t} R_i)$. (generate the hard constraint)
  3. For every tuple t over W_p do: (generate the cost function)
  $h^p(t) = max_{\{a_p|(t,a_p)\ satisfies\ \{R_1,...,R_l\}\}} \sum_{i=1}^{j} h_i(t, a_p)$.
  Place h^p in the latest lower bucket mentioning a variable in W_p. Place R^p in the bucket of the latest variable in U_p.

3. **Forward:** Assign maximizing values in ordering d, consulting functions in each bucket.

Figure 13.6: Algorithm elim-opt-cons

Bucket-elimination for counting

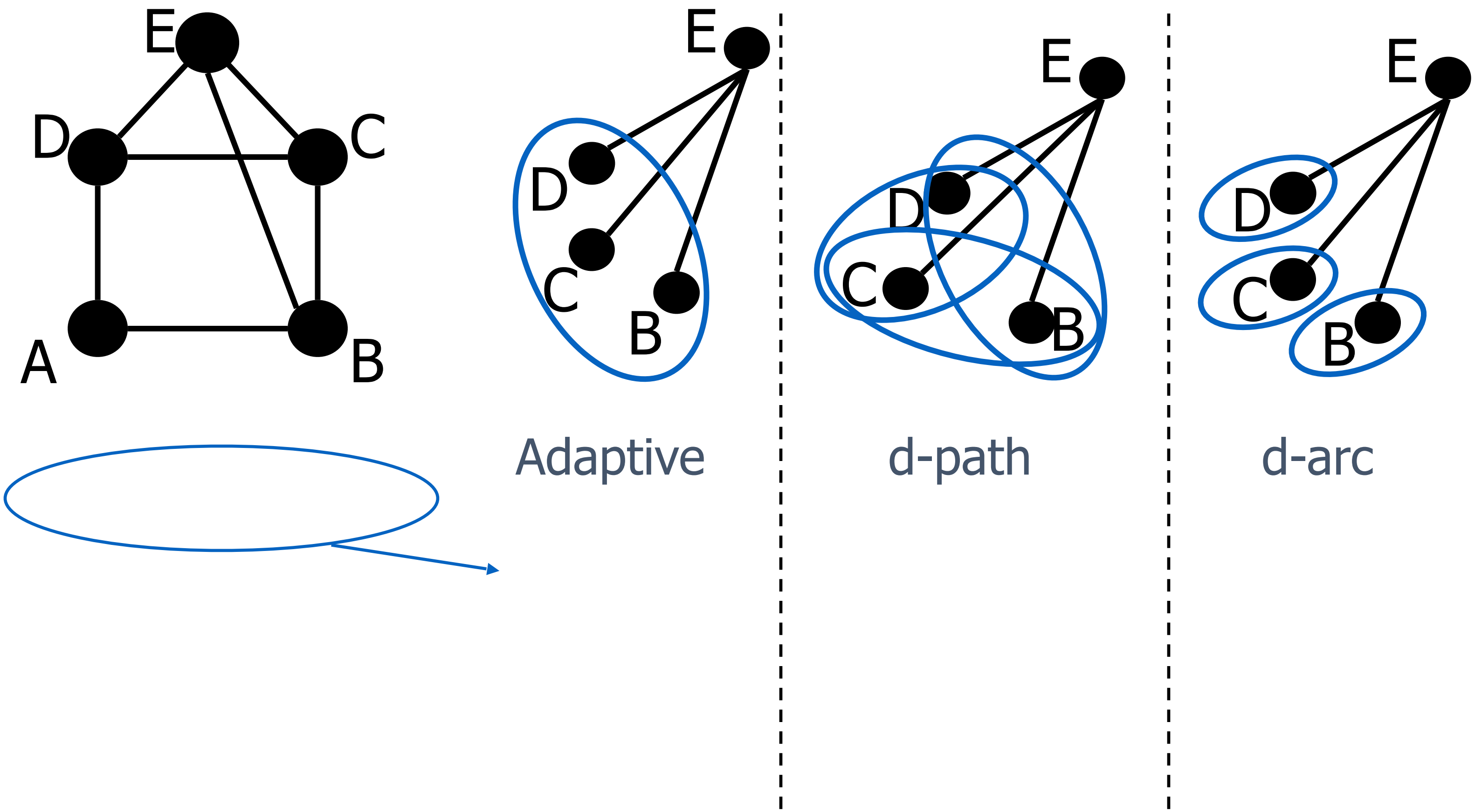
Algorithm elim-count
Input: A constraint network $\mathcal{R} = (X, D, C)$, ordering d.
Output: Augmented output buckets including the intermediate count functions and The number of solutions.
1. **Initialize:** Partition C (0-1 cost functions) into ordered buckets $bucket_1, \ldots, bucket_n$, We denote a function in a bucket N_i, and its scope S_i.)
2. **Backward:** For $p \leftarrow n$ downto 1, do
   Generate the function N^p: $N^p = \sum_{X_p} \prod_{N_i \in bucket_p} N_i$.
   Add N^p to the bucket of the latest variable in $\bigcup_{i=1}^{j} S_i - \{X_p\}$.
3. **Return** the number of solutions, N^1 and the set of output buckets with the original and computed functions.

Figure 13.9: Algorithm *elim-count*

Outline

- Introduction
  - Optimization tasks for graphical models
  - Solving by inference and search
- Inference
  - Bucket elimination, dynamic programming, tree-clustering, bucket-elimination
  - **Mini-bucket elimination, belief propagation**
- **Search**
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - AND/OR search spaces
- **Hybrids of search and inference**
  - Cutset decomposition
  - Super-bucket scheme

Directional i-consistency



Mini-Bucket Elimination (MAP/MPE)

Split a bucket into mini-buckets —> bound complexity

bucket (X) =

$$\{\; \underbrace{f_1,\; f_2,\; \ldots\; f_r,\; f_{r+1},\; \ldots\; f_n}\; \}$$

$$\lambda_X(\cdot) = \max_x \prod_{i=1}^{n} f_i(x, \ldots)$$

$$\{\; f_1,\; \ldots\; f_r \;\} \qquad\qquad \{\; f_{r+1},\; \ldots\; f_n \;\}$$

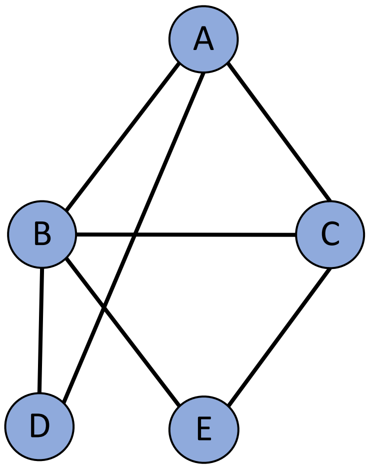
$$\lambda_{X,1}(\cdot) = \max_x \prod_{i=1}^{r} f_i(x, \ldots) \qquad\qquad \lambda_{X,2}(\cdot) = \max_x \prod_{i=r+1}^{n} f_i(x, \ldots)$$

$$\lambda_X(\cdot) \;\leq\; \lambda_{X,1}(\cdot)\; \lambda_{X,2}(\cdot)$$

Exponential complexity decrease: $O(e^n) \longrightarrow O(e^r) + O(e^{n-r})$

Mini-Bucket Elimination (MAP)

[Dechter & Rish 2003]



mini-buckets

bucket B: $\underbrace{f(a,b)\ f(b,c)}$ $\underbrace{f(b,d)\ f(b,e)}$

bucket C: $\underbrace{f(c,a)\ f(c,e)\ \lambda_{B\to C}(a,c)}$

bucket D: $\underbrace{f(a,d)\ \ \lambda_{B\to D}(d,e)}$

bucket E: $\underbrace{\lambda_{C\to E}(a,e)\ \lambda_{E\to E}(a,e)}$

bucket A: $\underbrace{f(a)\ \lambda_{E\to A}(a)}$

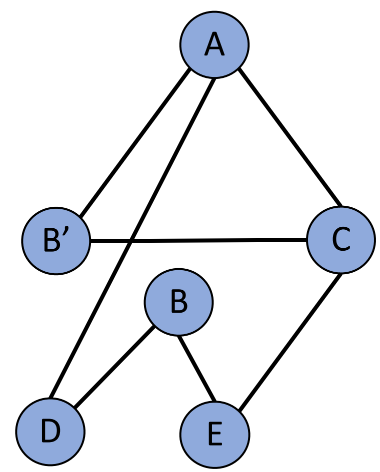
U = upper bound

$$\lambda_{B\to C}(a,c) = \max_b f(a,b)\ f(b,c)$$

$$\lambda_{B\to D}(d,e) = \max_b f(b,d)\ f(b,e)$$

$$\lambda_{C\to E}(a,e) = \max_c \ldots$$

In our case, min $\sum$



Mini-Bucket Decoding (MAP)

$$\mathbf{b}^* = \arg\max_b \; f(a^*, b) \cdot f(b, c^*) \cdot f(b, d^*) \cdot f(b, e^*)$$

$$\mathbf{c}^* = \arg\max_c \; f(c, a^*) \cdot f(c, e^*) \cdot \lambda_{B\to C}(a^*, c)$$

$$\mathbf{d}^* = \arg\max_d \; f(a^*, d) \cdot \lambda_{B\to D}(d, e^*)$$

$$\mathbf{e}^* = \arg\max_e \; \lambda_{C\to E}(a^*, e) \cdot \lambda_{D\to E}(a^*, e)$$

$$\mathbf{a}^* = \arg\max_a \; f(a) \cdot \lambda_{E\to A}(a)$$

mini-buckets

B: $f(a,b)\; f(b,c)$ $\quad$ $f(b,d)\; f(b,e)$

C: $f(c,a)\; f(c,e)\; \lambda_{B\to C}(a,c)$

D: $f(a,d)\; \lambda_{B\to D}(d,e)$

E: $\lambda_{C\to E}(a,e)\; \lambda_{E\to E}(a,e)$

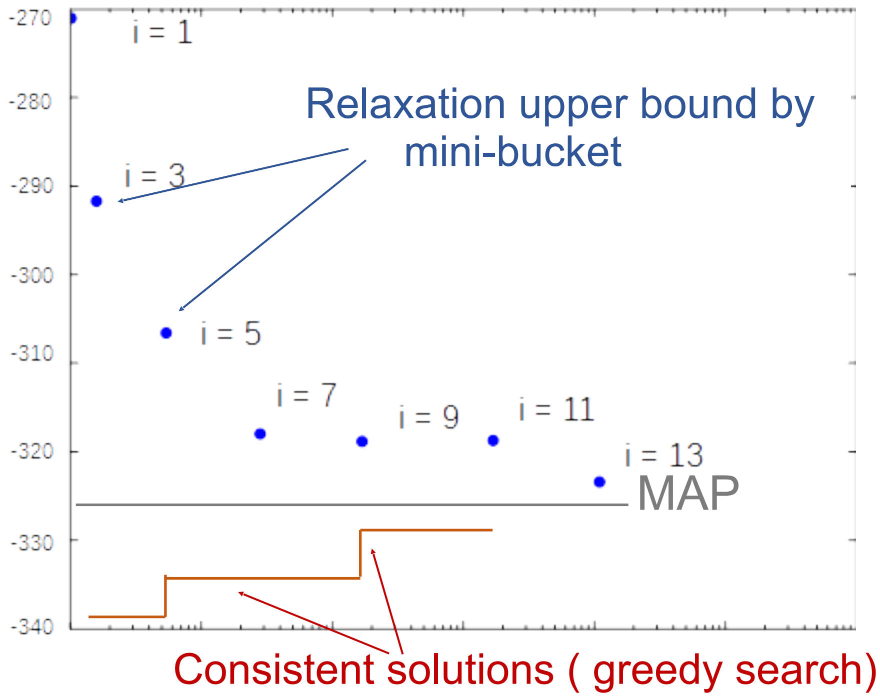
A: $f(a)\; \lambda_{E\to A}(a)$

U = upper bound

Greedy configuration = lower bound

Properties of Mini-Bucket Eliminaton

Bounding from above and below:



- **Complexity:** *O(r exp(i))* time and *O(exp(i)) space*.
- **Accuracy:** determined by Upper/Lower bound.
- As ***i*** increases, both accuracy and complexity increase.
- Possible use of mini-bucket approximations:
  - As anytime algorithms
  - As heuristics in search

MBE vs BE

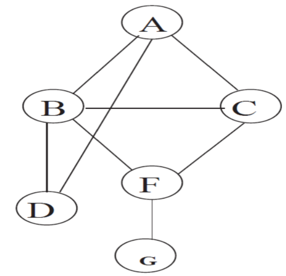
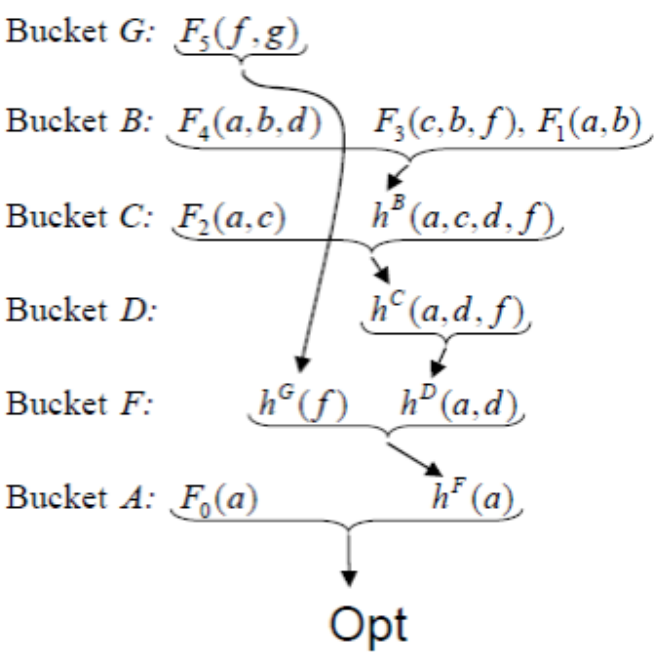
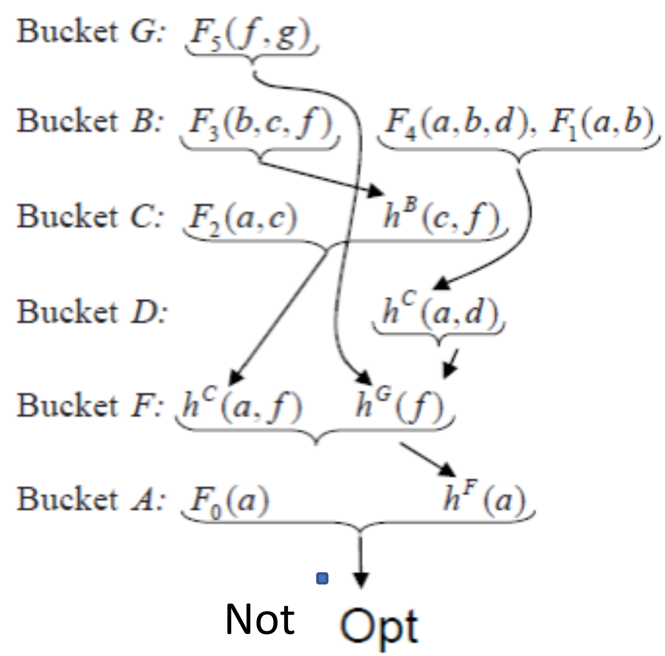


Figure 13.1: The cost graph of the cost function: $C(a,b,c,d,f,g) = F_0(a) + F_1(a,b) + F_2(a,c) + F_3(b,c,f) + F_4(a,b,d) + F_5(f,g)$



(a) A trace of *elim-opt*



(b) A trace of *mbe-opt(3)*

Bucket-elimination for combinatorial auction example

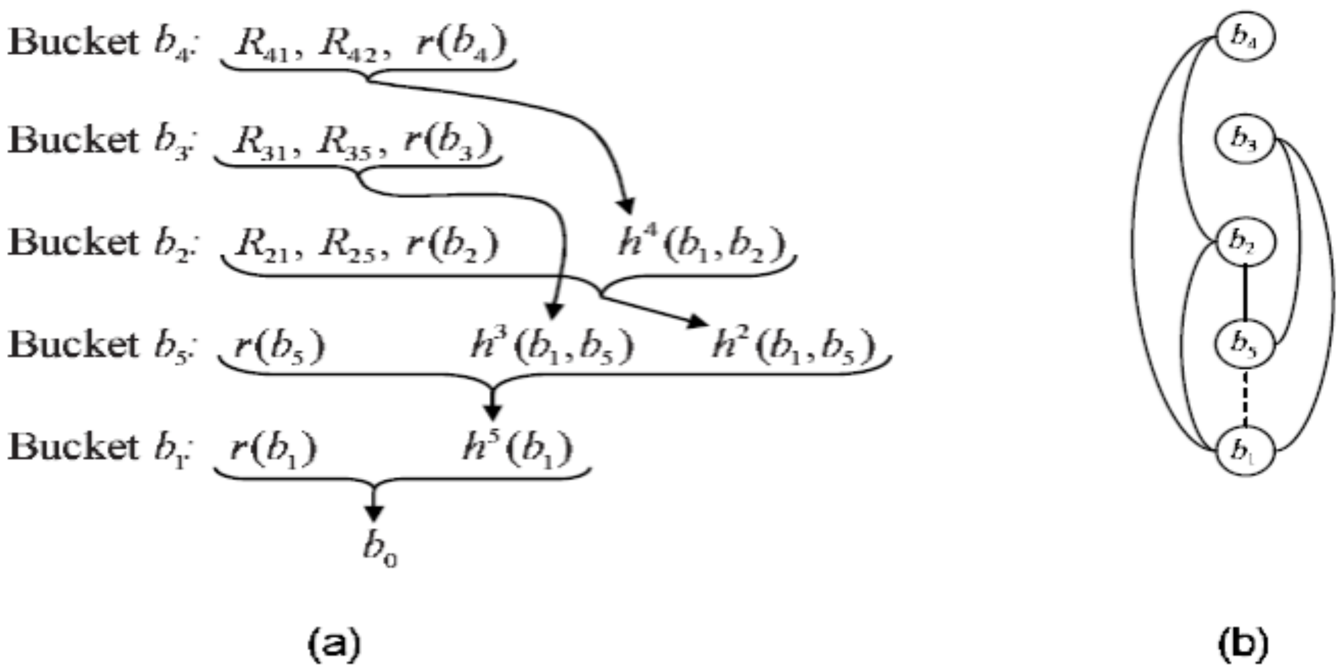


Figure 13.8: Schematic execution of elim-opt on the auction problem

Example mbe-opt

Example 13.4.3 Let us apply algorithm mbe-opt(2)to the auction problem along the ordering $d = b_1, b_5, b_2, b_3, b_4$. Figure 13.12 shows the resulting mini-buckets; square brackets denote the choice for partitioning.

We start with processing bucket b_4. A possible partitioning places the constraint R_{41} in one mini-bucket and the rest in the other. We compute a constraint $R^4(b_1) = \pi_{b_1} R_{41}$, which is the universal constraint so it need not be recorded. In the second mini-bucket, we also compute $h^4(b_2) = \max_{\{(b_4|(b_4,b_2)\in R_{42}\}} r(b_4)$, yielding:

$$h^4(b_2) = \begin{cases} 0 & \text{if } b_2 = 1 \\ 2 & \text{if } b_2 = 0 \end{cases}$$

Processing the first mini-bucket of b_3, which includes only hard constraints, will also not effect the domain of b_1. Processing the second mini-bucket of b_3 by $h^3(b_5) = \max_{\{(b_3|(b_3,b_5)\in R_{35}\}} r(b_3)$, yields:

$$h^3(b_5) = \begin{cases} 0 & \text{if } b_5 = 1 \\ 5 & \text{if } b_5 = 0 \end{cases}$$

Processing the second mini-bucket of b_2 (the first mini-bucket includes a constraint whose projection is a universal constraint) by $h^2(b_5) = \max_{\{(b_2|(b_2,b_5)\in R_{25}\}}(r(b_2) + h^4(b_2))$, gives us:

$$h^2(b_5) = \begin{cases} 2 & \text{if } b_5 = 1 \\ 6 & \text{if } b_5 = 0 \end{cases}$$

Processing bucket b_5 (with full buckets now) the algorithm computes $h^5 = \max_{b_5}(r(b_5) + h^2(b_5) + h^3(b_5))$, yielding:

$$h^5 = 11$$

Finally, in the bucket of b_1 the algorithm computes $h^1 = \max_{b_1}(r(b_1) + h^5)$, yielding: $h^1 = \max\{0 + 11, 8 + 11\} = 19$.

The maximal upper-bound cost is therefore 19. To compute a maximizing tuple we select in bucket b_1 the value $b_1 = 1$, which maximizes $r(b_1) + h^5$. Given $b_1 = 1$, we choose $b_5 = 0$, which maximizes the functions in bucket b_5. Then, in bucket b_2, we can choose only $b_2 = 0$, due to the constraint R_{12}. Likewise, the only subsequent choices are $b_3 = 0$ and $b_4 = 0$. Therefore, the cost of the generated solution is 8, yielding the interval [8,19] bounding the maximal solution. □

$$\begin{array}{ll}
\text{Bucket } b_4: & [R_{41}], [R_{42}, r(b_4)] \\
\text{Bucket } b_3: & [R_{31}], [R_{35}, r(b_3)] \\
\text{Bucket } b_2: & [R_{21}], [R_{25}, r(b_2) \;||\; h^4(b_2)] \\
\text{Bucket } b_5: & [r(b_5) \;||\; h^3(b_5), h^2(b_5)] \\
\text{Bucket } b_1: & r(b_1) \;||\; h^5 \\
\text{Yielding: opt:} & h^1 = M'
\end{array}$$

Algorithm mbe-opt

Algorithm mbe-opt(i)
Input: A cost network $\mathcal{C} = (X, D, C)$; an ordering d; parameter i.
Output: An upper bound on the optimal cost solution, a solution and a lower bound and the ordered augmented buckets.
1. **Initialize:** Partition the functions in C into $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all functions whose highest variable is x_i. Let $S_1, ..., S_j$ be the scopes of functions (new or old) in the processed bucket.
2. **Backward** For $p \leftarrow n$ down-to 1, do
• **If** variable x_p is instantiated ($x_p = a_p$), assign $x_p = a_p$ to each h_i and put each resulting function into its appropriate bucket.
• **Else,** for $h_1, h_2, ..., h_j$ in $bucket_p$, generate an (i)-partitioning, $Q' = \{Q_1, ..., Q_t\}$. For each $Q_l \in Q'$ containing $h_{l_1}, ...h_{l_t}$ generate function h^l, $h^l = \max_{x_p} \sum_{i=1}^{t} h_{l_i}$. Add h^l to the bucket of the largest-index variable in U_l, $U_l = \bigcup_{i=1}^{j} scope(h_{l_i}) - \{x_p\}$.
3. **Forward** For $i = 1$ to n do, given $a_1, ..., a_{p-1}$ choose a value a_p of x_p that maximizes the sum of all the functions in x_p's bucket.
4. **Return** the ordered set of augmented buckets, an assignment $\bar{a} = (a_1, ..., a_n)$, an interval bound (the value computed in $bucket_1$ and the cost $F(\bar{a})$).

Figure 13.9: Mini-Bucket Elimination Algorithm

Properties of MBE(i)

- **Complexity:** *O(r exp(i))* time and *O(exp(i)) space*.
- Yields an upper-bound and a lower-bound.

- **Accuracy:** determined by upper/lower (U/L) bound.

- As ***i*** increases, both accuracy and complexity increase.

- Possible use of mini-bucket approximations:
  - As anytime algorithms
  - As heuristics in search

- Other tasks: similar mini-bucket approximations for: belief updating, MAP and MEU (Dechter and Rish, 1997)

Outline

- **Introduction**
  - Optimization tasks for graphical models
  - Solving optimization problems with inference and search
- **Inference**
  - Bucket elimination, dynamic programming
  - Mini-bucket elimination
- **Search**
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - AND/OR search spaces

Branch and bound

procedure BRANCH-AND-BOUND
Input: A cost network $\mathcal{C} = (X, D, C_h, C_s)$, L current upper-bound, An upper-bound function f defined for every partial solution.
Output: Either an optimal (maximal) solution, or notification that the network is inconsistent.

$i \leftarrow 1$ (initialize variable counter)
$D'_i \leftarrow D_i$ (copy domain)
While $1 \leq i \leq n$
instantiate $x_i \leftarrow$ SELECTVALUE
If x_i is null (no value was returned)
$i \leftarrow i - 1$ (backtrack)
Else
$i \leftarrow i + 1$ (step forward)
$D'_i \leftarrow D_i$
Endwhile
If $i = 0$
Return "inconsistent"
Else
Compute $C = C(x_1, ...x_n)$, $U \leftarrow max\{C, L\}$
$i \leftarrow n - 1$
end procedure

procedure SELECTVALUE
If $i = 0$ return U as the solution value and the most recent assignment as solution.
While D'_i is not empty
select an element $a \in D'_i$ having max $f(\vec{a}_{i-1}, a)$
and remove a from D'_i
If $< x_i, a >$ is consistent with $\vec{a}_{i-1}$ and $f > L$, then
Return a (else prune a)

Endwhile
Return null (no consistent value)
end procedure

Figure 13.2: The Branch and Bound algorithm

Search tree for first-choice heuristic

$$f_{fc}(\vec{a}_i) = \sum_{F_j \in C} \max_{a_{i+1},\dots a_n} F_j(\vec{a}_n)$$

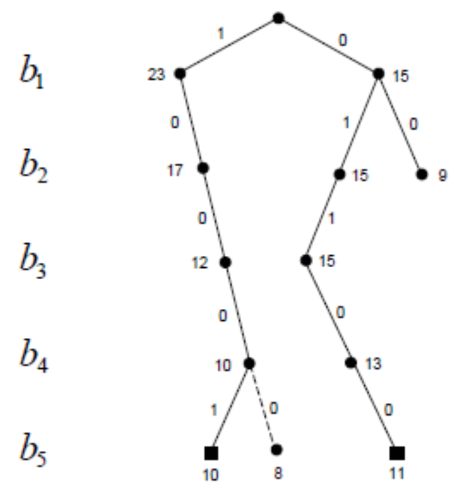
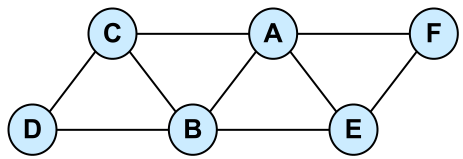


Figure 13.3: Branch and bound search space for the auction problem

Example 13.2.1 Consider the auction problem described in Example 13.1.2. Searching for a solution in the order $d = b_1, b_2, b_3, b_4, b_5$ yields the search space in Figure 13.3, traversed from left to right. The search space is highly constrained in this case. The evaluation bounding function at each node should *overestimates* its best extension for a solution. The first-choice bounding function for the root node (the empty assignment) is 23. The first solution encountered (selecting $< b_1, 1 >$ and $< b_5, 1 >$) has a cost of 10, which becomes the current global *lower bound*. The next solution encountered has the cost of 11 (when $< b_2, 1 >$ and $< b_3, 1 >$), while the rest of the variables are assigned 0). Subsequently, the partial assignment $(< b_1, 0 >, < b_2, 0 >)$ is explored. Since $f_{fc}(< b_1, 0 >, < b_2, 0 >) = 9$, this upper bound is lower than 11, and therefore search can be pruned. □

The search space



A	B	f_1
0	0	**2**
0	1	**0**
1	0	**1**
1	1	**4**

A	C	f_2
0	0	**3**
0	1	**0**
1	0	**0**
1	1	**1**

A	E	f_3
0	0	**0**
0	1	**3**
1	0	**2**
1	1	**0**

A	F	f_4
0	0	**2**
0	1	**0**
1	0	**0**
1	1	**2**

B	C	f_5
0	0	**0**
0	1	**1**
1	0	**2**
1	1	**4**

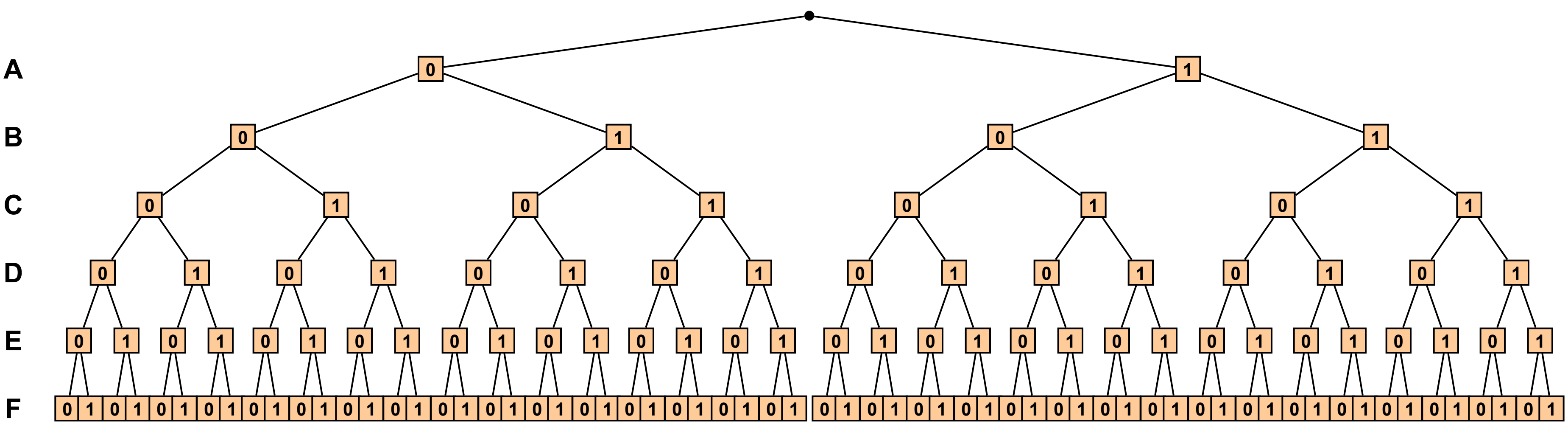
B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

B	E	f_7
0	0	**3**
0	1	**2**
1	0	**1**
1	1	**0**

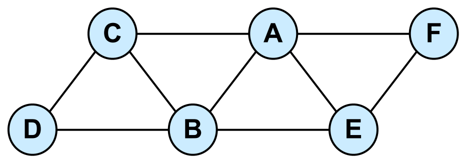
C	D	f_8
0	0	**1**
0	1	**4**
1	0	**0**
1	1	**0**

E	F	f_9
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**

Objective function:



The search space



A	B	f_1
0	0	**2**
0	1	**0**
1	0	**1**
1	1	**4**

A	C	f_2
0	0	**3**
0	1	**0**
1	0	**0**
1	1	**1**

A	E	f_3
0	0	**0**
0	1	**3**
1	0	**2**
1	1	**0**

A	F	f_4
0	0	**2**
0	1	**0**
1	0	**0**
1	1	**2**

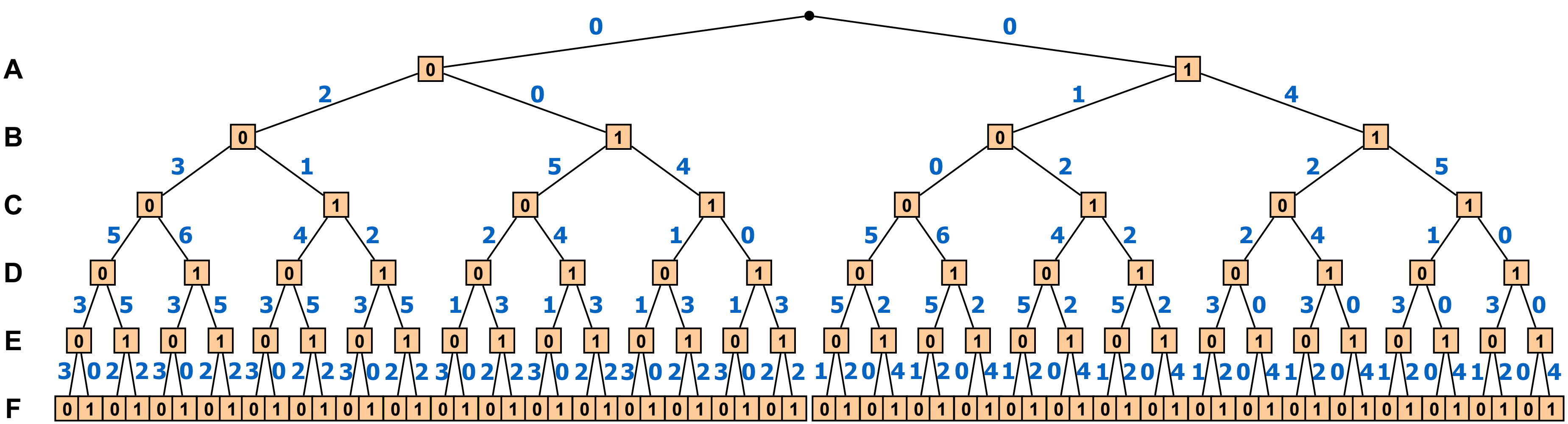
B	C	f_5
0	0	**0**
0	1	**1**
1	0	**2**
1	1	**4**

B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

B	E	f_7
0	0	**3**
0	1	**2**
1	0	**1**
1	1	**0**

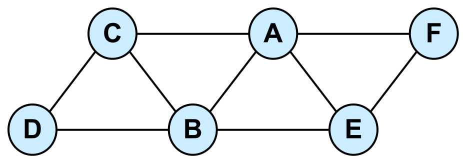
C	D	f_8
0	0	**1**
0	1	**4**
1	0	**0**
1	1	**0**

E	F	f_9
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**



Arc-cost is calculated based on cost components.

The value function



A	B	f_1
0	0	**2**
0	1	**0**
1	0	**1**
1	1	**4**

A	C	f_2
0	0	**3**
0	1	**0**
1	0	**0**
1	1	**1**

A	E	f_3
0	0	**0**
0	1	**3**
1	0	**2**
1	1	**0**

A	F	f_4
0	0	**2**
0	1	**0**
1	0	**0**
1	1	**2**

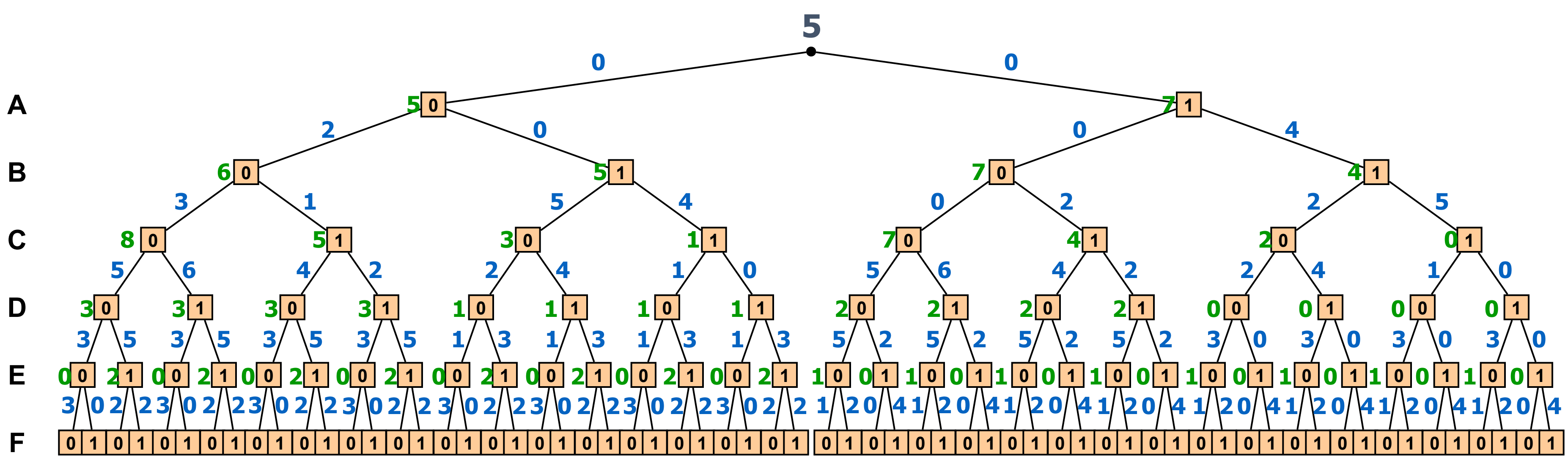
B	C	f_5
0	0	**0**
0	1	**1**
1	0	**2**
1	1	**4**

B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

B	E	f_7
0	0	**3**
0	1	**2**
1	0	**1**
1	1	**0**

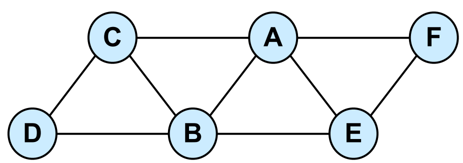
C	D	f_8
0	0	**1**
0	1	**4**
1	0	**0**
1	1	**0**

E	F	f_9
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**



Value of node = minimal cost solution below it

An optimal solution



A	B	f_1
0	0	**2**
0	1	**0**
1	0	**1**
1	1	**4**

A	C	f_2
0	0	**3**
0	1	**0**
1	0	**0**
1	1	**1**

A	E	f_3
0	0	**0**
0	1	**3**
1	0	**2**
1	1	**0**

A	F	f_4
0	0	**2**
0	1	**0**
1	0	**0**
1	1	**2**

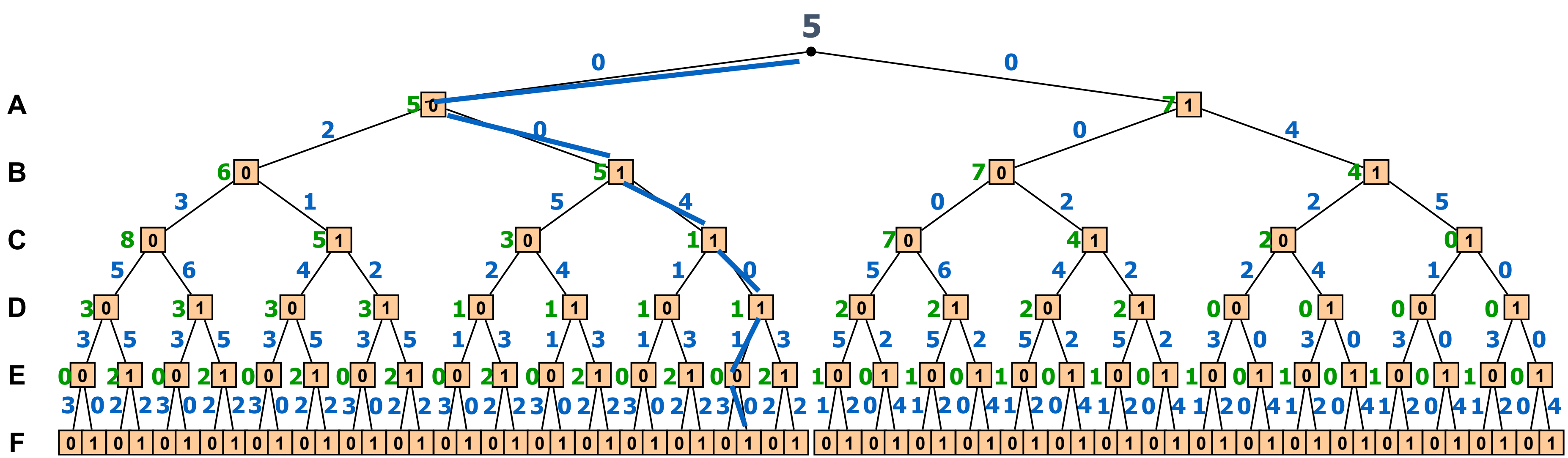
B	C	f_5
0	0	**0**
0	1	**1**
1	0	**2**
1	1	**4**

B	D	f_6
0	0	**4**
0	1	**2**
1	0	**1**
1	1	**0**

B	E	f_7
0	0	**3**
0	1	**2**
1	0	**1**
1	1	**0**

C	D	f_8
0	0	**1**
0	1	**4**
1	0	**0**
1	1	**0**

E	F	f_9
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**2**

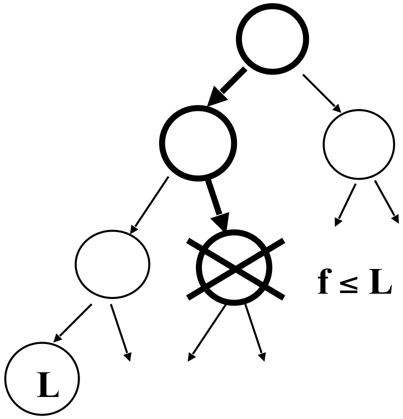


Value of node = minimal cost solution below it

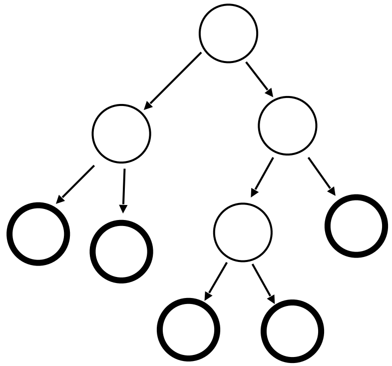
Basic heuristic search schemes

Heuristic function f(x) computes a lower bound on the best extension of x and can be used to guide a heuristic search algorithm. We focus on

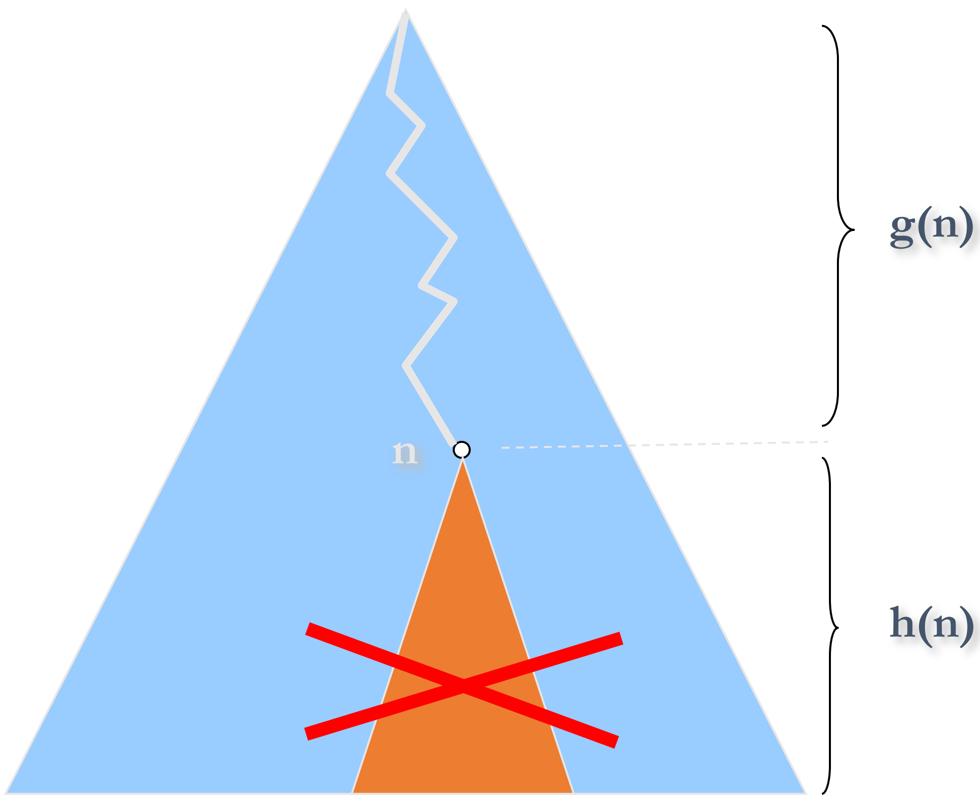
1.Branch and Bound
Use heuristic function $f(x^p)$ to prune the depth-first search tree.
Linear space



2.Best-First Search
Always expand the node with the highest heuristic value $f(x^p)$.
Needs lots of memory



Classic branch-and-bound



OR Search Tree

Upper Bound **UB**

Lower Bound **LB**

$LB(n) = g(n) + h(n)$

Prune if $LB(n) \geq UB$

How to Generate Heuristics

- The principle of relaxed models
  - Linear optimization for integer programs
  - Mini-bucket elimination
  - Bounded directional consistency ideas

Generating Heuristics for Graphical Models

Given a cost function:

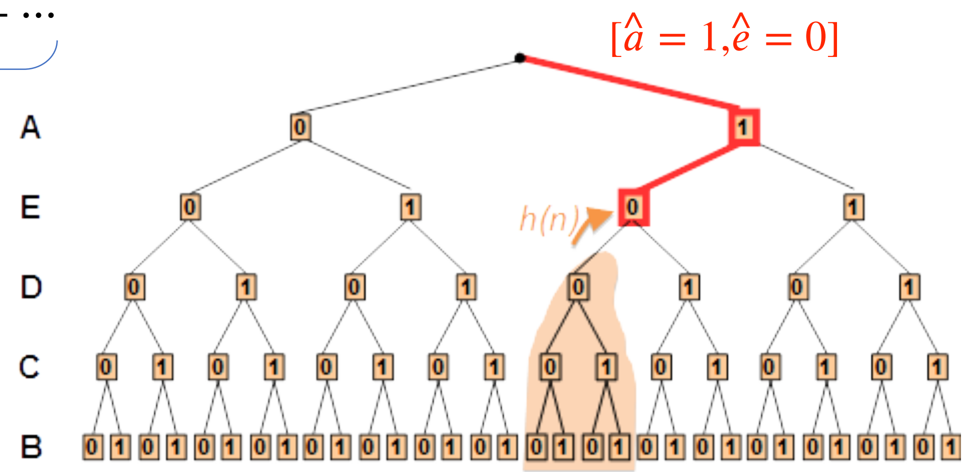
$$f(a,\ \ldots,\ e) = f(a) + f(a,b) + f(a,c) + f(a,d) + f(b,c) + f(b,d) + f(b,e) + f(c,e)$$

define an evaluation function over a partial assignment as the cost of its best extension:

$$f^*\left(\hat{a},\hat{e},D\right) = \min_{b,c} F(\hat{a},b,c,D,\hat{e})$$

$$= f\left(\hat{a}\right) + \min_{b,c} f\left(\hat{a},b\right) + f\left(\hat{a},c\right) + \cdots$$

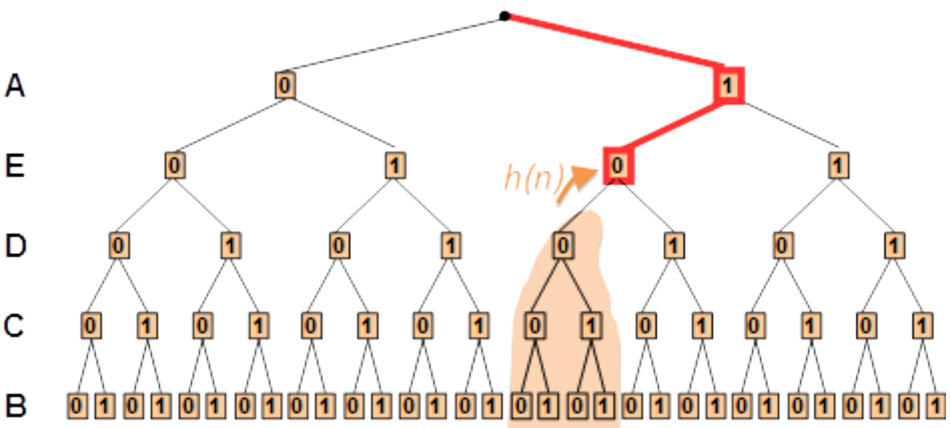
$$= g\left(\hat{a},\hat{e},D\right) + h^*(\hat{a},\hat{e},D)$$



[Kask and Dechter, 2001]

Static Mini-Bucket Heuristics

Given a partial assignment, $[\hat{a} = 1, \hat{e} = 0]$
mini-bucket gives an admissible heuristic:



mini-buckets

$\min_B \sum f(\cdot)$

bucket B: $f(a,b) \quad f(b,c) \qquad f(b,d) \quad f(b,e)$

bucket C: $\lambda_{B\to C}(a,c) \quad f(c,a) \quad f(c,e)$

bucket D: $f(a,d) \quad \lambda_{B\to D}(d,e)$

bucket E: $\lambda_{C\to E}(a,e) \quad \lambda_{D\to E}(a,e)$

bucket A: $f(a) \quad \lambda_{E\to A}(a)$

L = lower bound

cost to go:

$$\tilde{h}\left(\hat{a}, \hat{e}, D\right) = \lambda_{C\to E}(\hat{a}, \hat{e}) + f\left(\hat{a}, D\right) + \lambda_{B\to D}(D, \hat{e})$$

(admissible: $\tilde{h}\left(\hat{a}, \hat{e}, D\right) \le h^*(\hat{a}, \hat{e}, D)$)

cost so far:

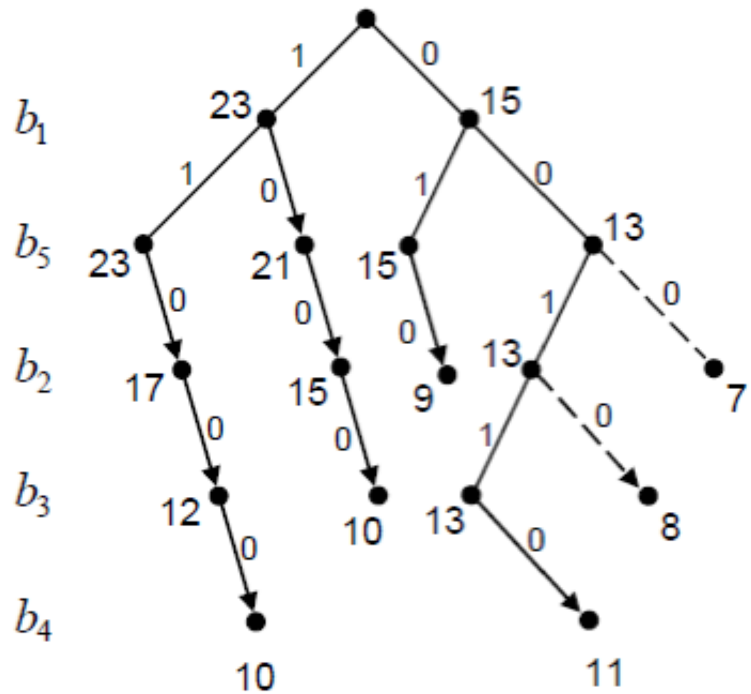
$$g\left(\hat{a}, \hat{e}, D\right) = f(A = \hat{a})$$

BBMB(i) with mini-bucket heuristics

$b1 = \{1;2;3;4\}$; $b2 = \{2;3;6\}$;
$b3 = \{1;5;4\}$ $b4 = \{2;8\}$ $b5 = \{5;6\}$
costs $r1 = 8$; $r2 = 6$; $r3 = 5$; $r4 = 2$; $r5 = 2$

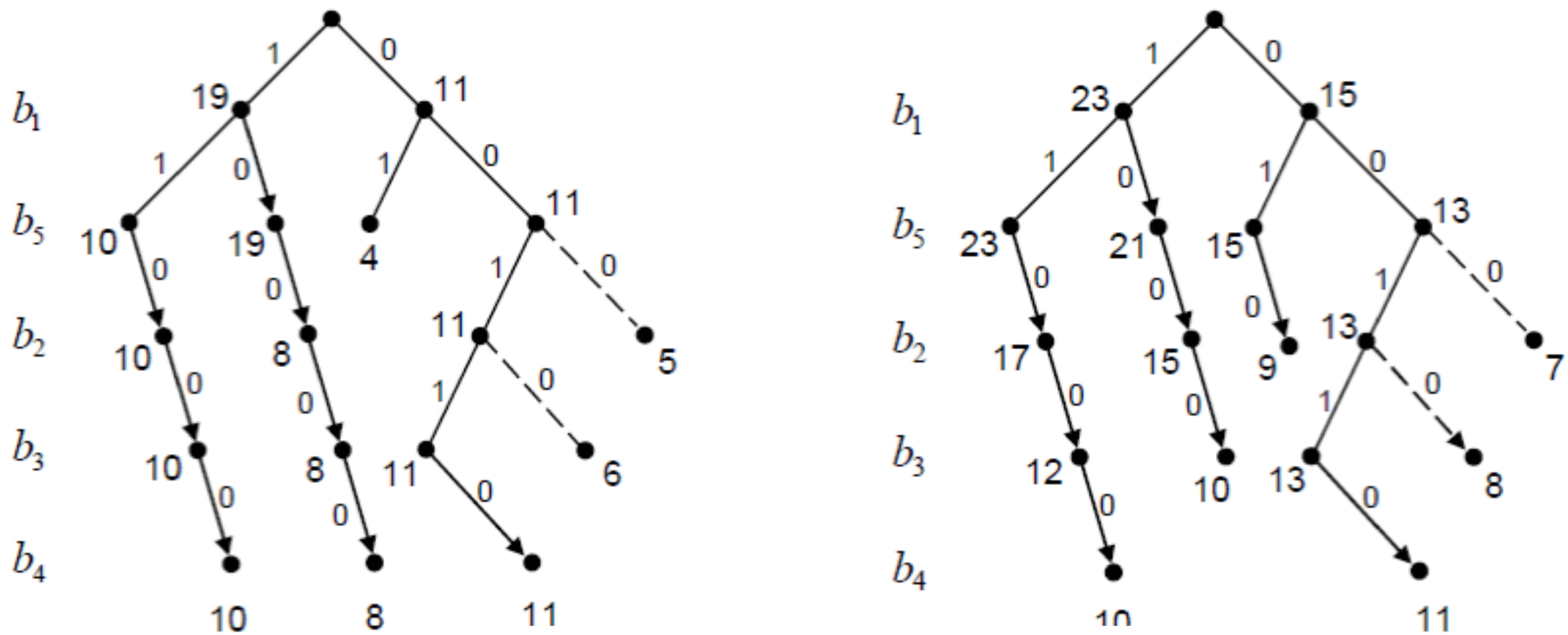
Let us now apply BnB with f_{mb}, which is the bounding function extracted from mbe-opt(2). Based on the functions in the augmented bucket of b_1 produced by mbe-opt(2), $f_{mb}(b_1 = 0) = r(b_1) + h^5 = 8 + 11 = 19$ while $f_{mb}(b_1 = 1) = 0 + 11 = 11$. Consequently, $b_1 = 1$ is chosen. We next evaluate $f_{mb}(b_1 = 1, b_5) = 8 + h^3(b_5) + h^2(b_5)$, yielding $f_{mb}(b_5 = 0) = 19$ and $f_{mb}(b_5 = 1) = 10$. Consequently, $b_5 = 0$ is chosen. The path is now deterministic, allowing the only choices: $b_2 = b_3 = b_4 = 0$. We end up with a solution having a cost of 8, which becomes the first global lower bound $L = 8$. *BnB* backtracks. The path is deterministic, dictating the choices ($b_2 = b_3 = b_4 = 0$) whose bounding cost equals 10, yielding a solution with cost 10 ($b_1 = 1, b_5 = 1, b_2 = 0, b_3 = 0, b_4 = 0,$). The global lower bound L is updated to 10. The algorithm backtracks to the next choice point, which is ($b_1 = 0$), whose bounding cost is 11. Next, for b_5, $f_{mb}(b_1 = 0, b_5) = 0 + r(b_5) + h^2(b_5) + h^3(b_5)$, yielding $f_{mb}(b_5 = 1) = 4$, which can be immediately pruned (less than 10), and $f_{mb}(b_5 = 1) = 11$. We select $b_5 = 0$. Subsequently, choosing a value for b_2 we get: $f_{mb}(b_1 = 0, b_5 = 0, b_2) = r(b_2) + h^4(b_2) + h^3(b_5)$ (note that $h^2(b_5)$ is not included since it is created in bucket b_2). This yields $f_{mb}(b_2 = 1) = 11$, while $f_{mb}(b_2 = 0) = 5$, which is pruned. The next choice for b_3 is determined using the bounding function $f_{mb}(b_1 = 0, b_5 = 0, b_2 = 1, b_3) = 6 + r(b_3) + h^4(b_2 = 1)$, yielding for $b_3 = 1$ the bound 11, while for $b_3 = 0$ the bound 6, which will be pruned. $b_3 = 1$ is selected. Subsequently, only $b_4 = 0$ is feasible and we get the best cost solution of value 11. The global lower bound, L is updated to 11 and *BnB* will lead to only pruned choices.

We see in this example that the performance of *BnB* using these two bounding func-



Bucket b_4 : $[R_{41}], [R_{42}, r(b_4)]$
Bucket b_3 : $[R_{31}], [R_{35}, r(b_3)]$
Bucket b_2 : $[R_{21}], [R_{25}, r(b_2) \;||\; h^4(b_2)]$
Bucket b_5 : $[r(b_5) \;||\; h^3(b_5), h^2(b_5)]$
Bucket b_1 : $r(b_1) \;||\; h^5$
Yielding: opt: $h^1 = M'$

BnB with first-cut (b) and mini-bucket (a) heuristics (BBMB(i))



Bucket b_4 : $[R_{41}], [R_{42}, r(b_4)]$
Bucket b_3 : $[R_{31}], [R_{35}, r(b_3)]$
Bucket b_2 : $[R_{21}], [R_{25}, r(b_2) \;\; || \;\; h^4(b_2)]$
Bucket b_5 : $[r(b_5) \;\; || \;\; h^3(b_5), h^2(b_5)]$
Bucket b_1 : $r(b_1) \;\; || \; h^5$
Yielding: opt: $h^1 = M'$

(a) b)

Heuristics properties

- MBE Heuristic is monotone, admissible
- Retrieved in linear time
- IMPORTANT:
  - Heuristic strength can vary by MBE(i).
  - Higher i-bound $\Rightarrow$ more pre-processing $\Rightarrow$ stronger heuristic $\Rightarrow$ less search.
- Allows controlled trade-off between preprocessing and search

Experimental methodology

Algorithms

- BBMB(i) – Branch and Bound with MB(i)
- BBFB(i) - Best-First with MB(i)
- MBE(i)

Test networks:

- Random Coding (Bayesian)
- CPCS (Bayesian)
- Random (CSP)

Measures of performance

- Compare accuracy given a fixed amount of time - how close is the cost found to the optimal solution
- Compare trade-off performance as a function of time

Empirical evaluation of mini-bucket heuristics, Bayesian networks, coding

Random Coding, K=100, noise=0.28

Random Coding, K=100, noise=0.32

Max-CSP experiments

(Kask and Dechter, 2000)

T	MBE BBMB BFMB i=2 #/time	MBE BBMB BFMB i=4 #/time	MBE BBMB BFMB i=6 #/time	MBE BBMB BFMB i=8 #/time	MBE BBMB BFMB i=10 #/time	MBE BBMB BFMB i=12 #/time	PFC-MRDAC #/time
N=100, K=3, C=200. Time bound 1 hr. Avg w^*=21. Sparse network.							
1	70/0.03 90/12.5 80/0.03	90/0.06 **100/0.07** **100/0.07**	100/0.32 100/0.33 100/0.33	100/2.15 100/2.16 100/2.15	100/15.1 100/15.1 100/15.1	100/116 100/116 100/116	100/0.08
2	0/- 0/- 0/-	0/- 0/- 0/-	4/0.35 96/644 56/131	20/2.28 **92/41** 88/170	20/15.6 96/69 92/135	24/123 100/125 100/130	100/757
3	0/- 0/- 0/-	0/- 0/- 0/-	0/- 100/996 16/597	0/- 100/326 60/462	4/14.4 **100/94.6** 88/344	4/114 100/190 84/216	100/2879
4	0/- 0/- 0/-	0/- 0/- 0/-	0/- 52/2228 4/2934	0/- 88/1042 8/540	4/14.9 92/396 28/365	8/120 **100/283** 60/866	100/7320

Dynamic mbe heuristics

- Rather than pre-compiling, the mini-bucket heuristics can be generated during search
- D*ynamic mini-bucket heuristics* use the Mini-Bucket algorithm to produce a bound for any node in the search space

(a partial assignment, along the given variable ordering)

Branch and bound w/ mini-buckets

- BB with static Mini-Bucket Heuristics (s-BBMB)
  - Heuristic information is pre-compiled before search. Static variable ordering, prunes current variable
- BB with dynamic Mini-Bucket Heuristics (d-BBMB)
  - Heuristic information is assembled during search. Static variable ordering, prunes current variable
- BB with dynamic Mini-Bucket-Tree Heuristics (BBBT)
  - Heuristic information is assembled during search. Dynamic variable ordering, prunes all future variables

Empirical evaluation

- Algorithms:
  - Complete
    - BBBT
    - BBMB
  - Incomplete
    - DLM
    - GLS
    - SLS
    - IJGP
    - IBP (coding)

- Measures:
  - Time
  - Accuracy (% exact)
  - #Backtracks
  - Bit Error Rate (coding)

- Benchmarks:
  - Coding networks
  - Bayesian Network Repository
  - Grid networks (N-by-N)
  - Random noisy-OR networks
  - Random networks

Real World Benchmarks

Network	# vars	avg. dom.	max dom.	BBBT/ BBMB/ IJGP i=2 %[time]	BBBT/ BBMB/ IJGP i=4 %[time]	BBBT/ BBMB/ IJGP i=6 %[time]	BBBT/ BBMB/ IJGP i=8 %[time]	GLS % [time]	DLM % [time]	SLS % [time]
Mildew	35	17	100	**100[0.28]** 30[10.5] 90[3.59]	**100[0.56]** 95[0.18] 97[33.3]	- - -	- - -	15 [30.02]	0 [30.02]	90 [30.02]
Munin2	1003	5	21	95[1.65] 95[30.3] 95[2.44]	95[1.65] 95[30.5] 95[5.17]	95[2.32] 95[31.3] 95[64.9]	**100[1.97]** **100[1.84]** -	0 [30.01]	0 [30.01]	0 [30.01]
Pigs	441	3	3	**90[15.2]** 0[30.01] 80[0.31]	**100[3.73]** 60[4.85] 77[0.53]	**100[2.36]** 80[0.02] 80[1.43]	**100[0.58]** 95[0.04] 83[6.27]	10 [30.02]	0 [30.02]	0 [30.02]
CPCS360b	360	2	2	100[0.17] **100[0.04]** 100[10.6]	100[0.27] **100[0.03]** 100[10.5]	100[0.21] **100[0.03]** 100[9.82]	100[0.19] **100[0.03]** 100[8.59]	100 [30.02]	100 [30.02]	100 [30.02]

Average Accuracy and Time. 30 samples, 10 observations, 30 seconds

Empirical Results: Max-CSP

- **Random Binary Problems**: <N, K, C, T>
  - N: number of variables
  - K: domain size
  - C: number of constraints
  - T: Tightness

- **Task**: Max-CSP

BBBT(i) vs BBMB(i), N=100

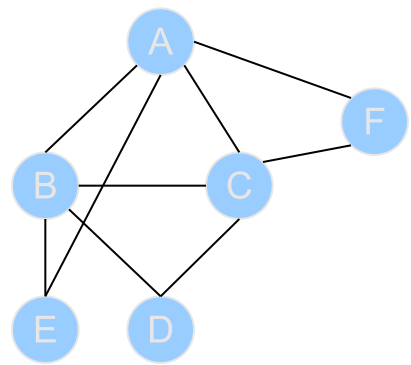
$N = 100,\ K = 5,\ C = 300.\ w^* = 33.9.$ 10 instances. time = 600sec.								
T	BBMB i=2	BBMB i=3	BBMB i=4	BBMB i=5	BBMB i=6	BBMB i=7	BBBT i=2	PFC-MPRDAC
	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks	# solved time backtracks
3	6 6 150K	6 6 150K	6 6 150K	6 5 115K	8 6.8 115K	8 15 8	10 7.73 60	10 0.03 750
5	2 36 980K	2 32 880K	2 24 650K	2 5.3 130K	3 38 870K	3 33 434K	10 14.3 114	10 0.06 1.5K
7	0	0	0	0	0	0	10 29 331	6 267 1.6M

BBBT(i) vs. BBMB(i).

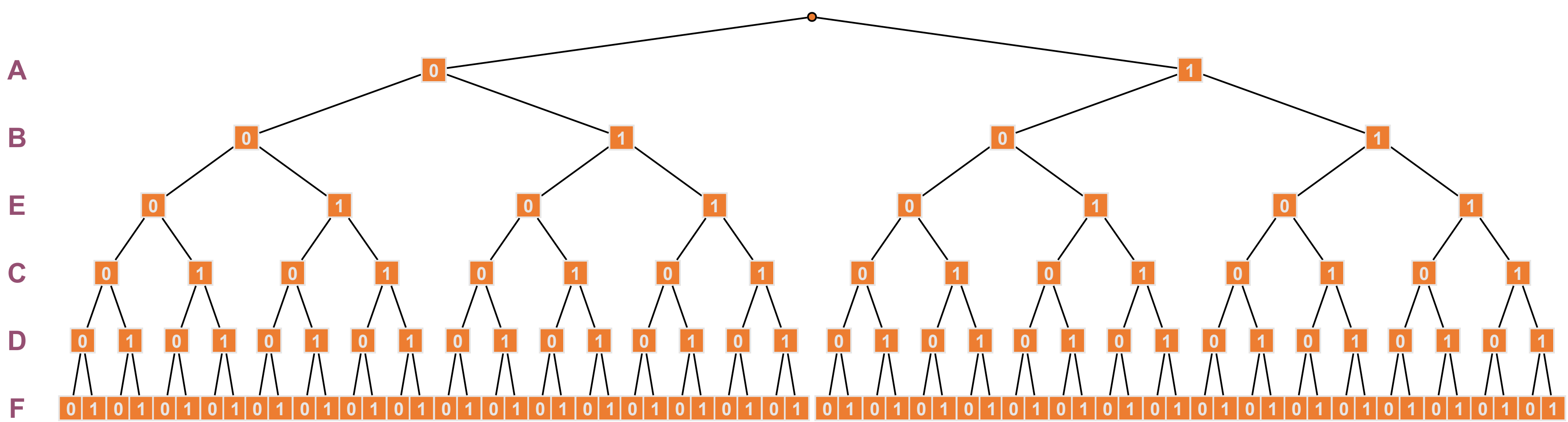
Outline

- **Introduction**
  - Optimization tasks for graphical models
  - Solving optimization problems with inference and search
- **Inference**
  - Bucket elimination, dynamic programming
  - Mini-bucket elimination
- **Search**
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - AND/OR search spaces

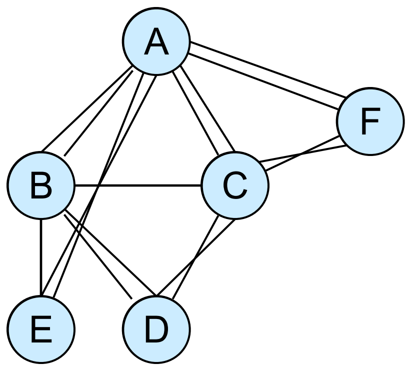
Classic OR search space



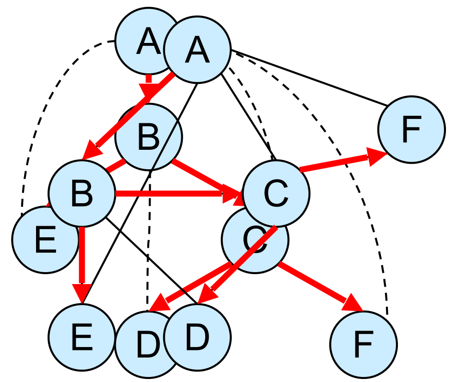
Ordering: A B E C D F



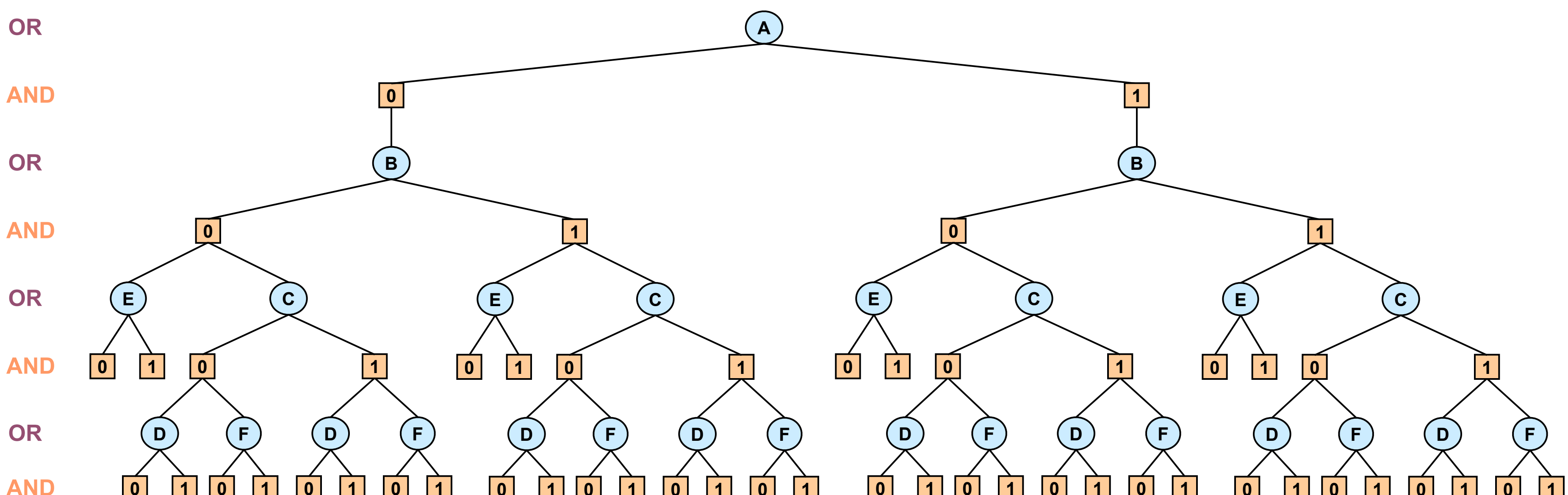
AND/OR search space



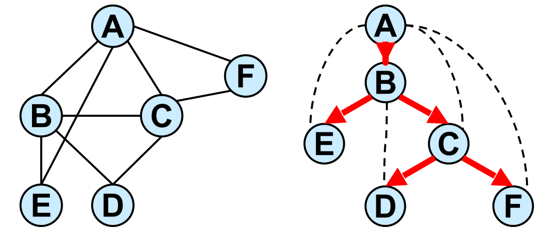
Primal graph

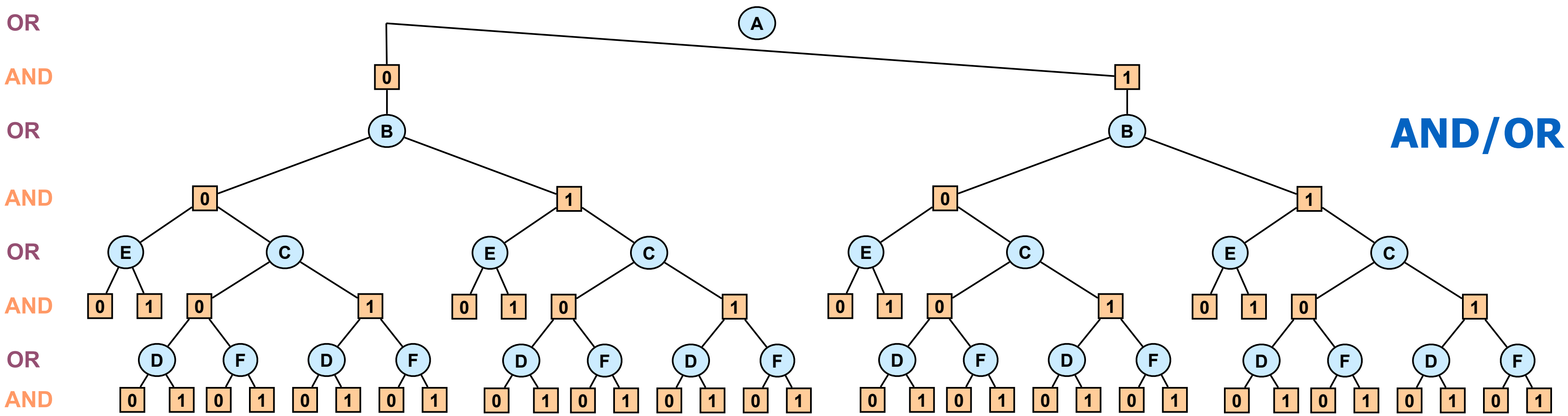


DFS tree

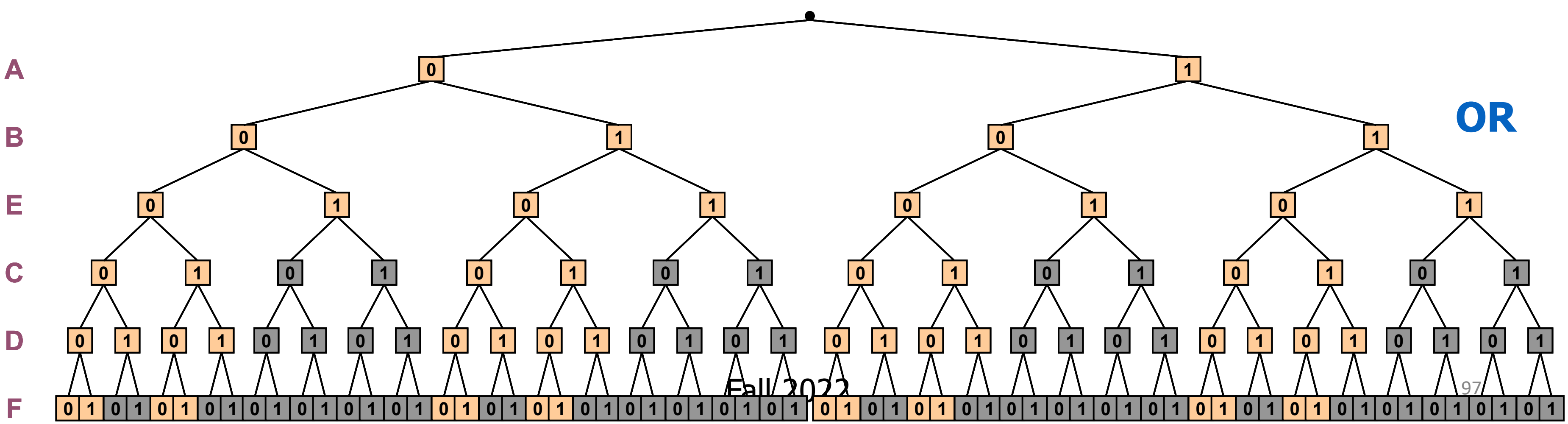


AND/OR vs. OR





AND/OR size: exp(4), OR size exp(6)



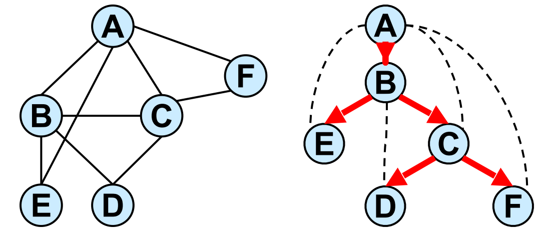
OR space vs. AND/OR space

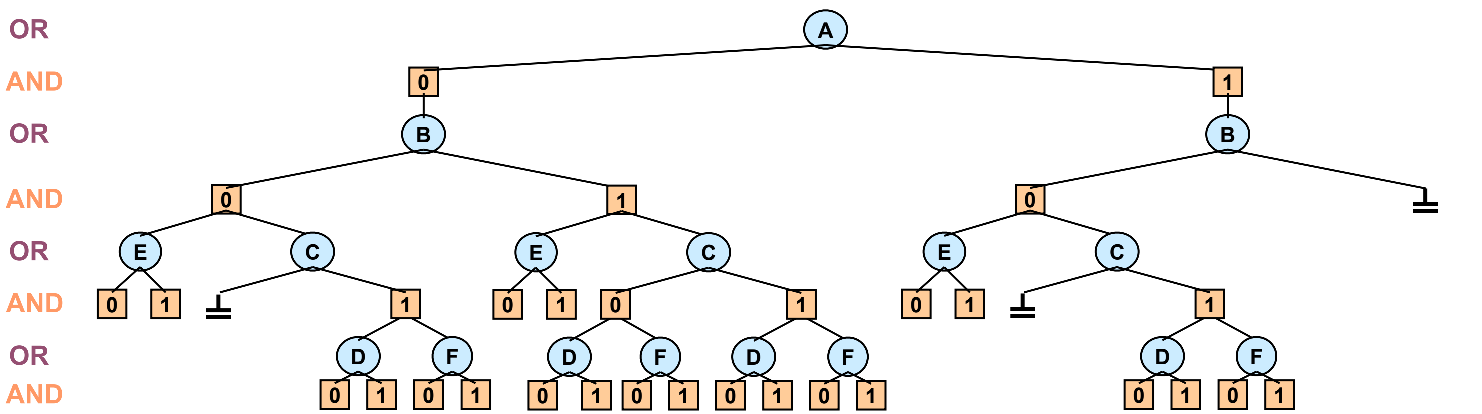
width	height	OR space			AND/OR space		
		Time (sec.)	Nodes	Backtracks	Time (sec.)	AND nodes	OR nodes
5	10	3.154	2,097,150	1,048,575	0.03	10,494	5,247
4	9	3.135	2,097,150	1,048,575	0.01	5,102	2,551
5	10	3.124	2,097,150	1,048,575	0.03	8,926	4,463
4	10	3.125	2,097,150	1,048,575	0.02	7,806	3,903
5	13	3.104	2,097,150	1,048,575	0.1	36,510	18,255

Random graphs with 20 nodes, 20 edges and 2 values per node.

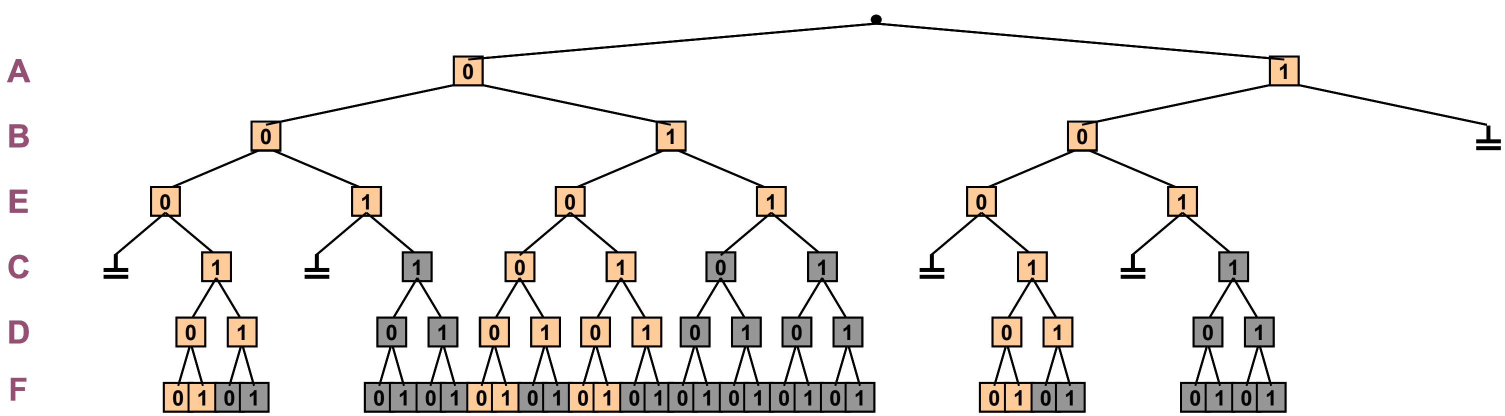
AND/OR vs. OR

(A=1,B=1)
(B=0,C=0)





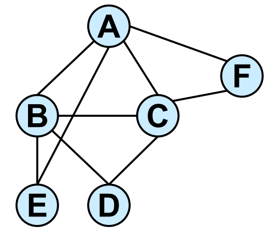
AND/OR

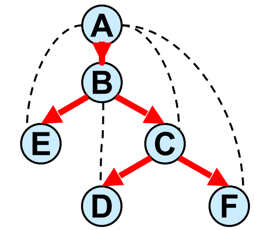


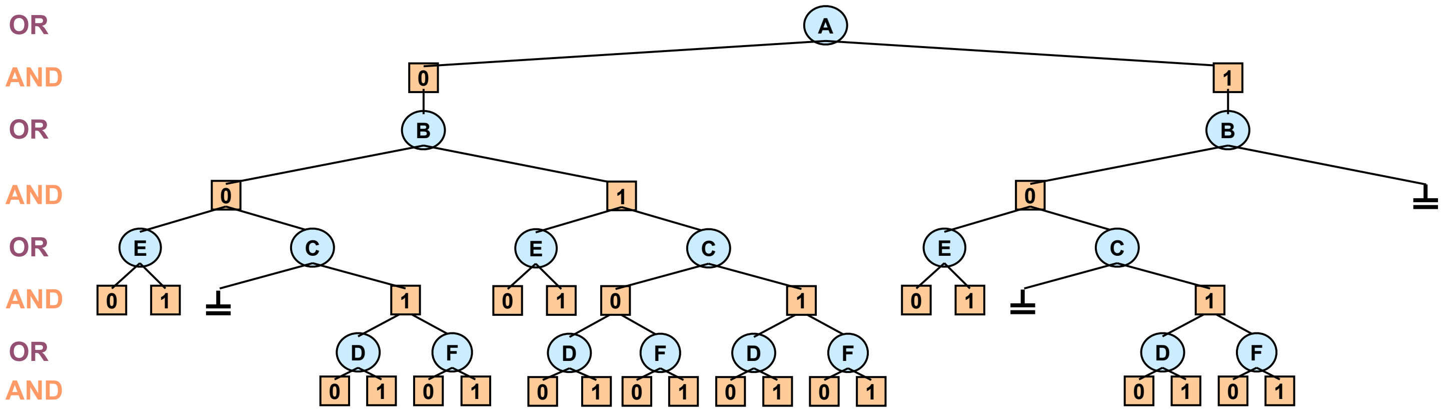
OR

AND/OR vs. OR

(A=1,B=1)
(B=0,C=0)

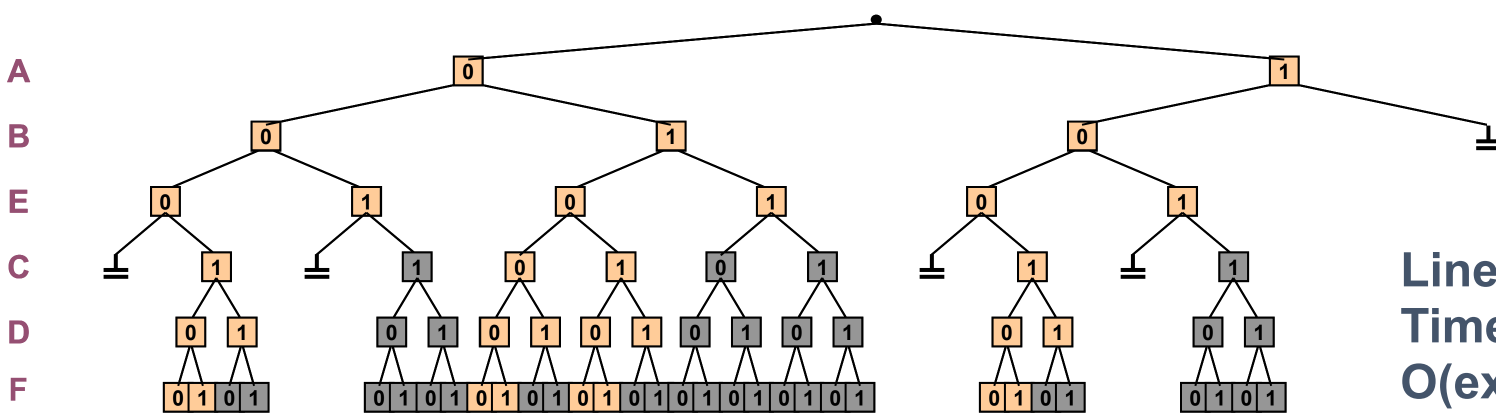






AND/OR

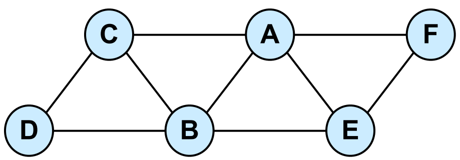
Space: linear
Time:
O(exp(m))
O(w* log n)



OR

Linear space,
Time:
O(exp(n))

#CSP – AND/OR search tree

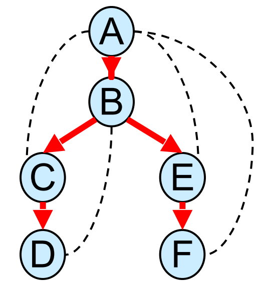


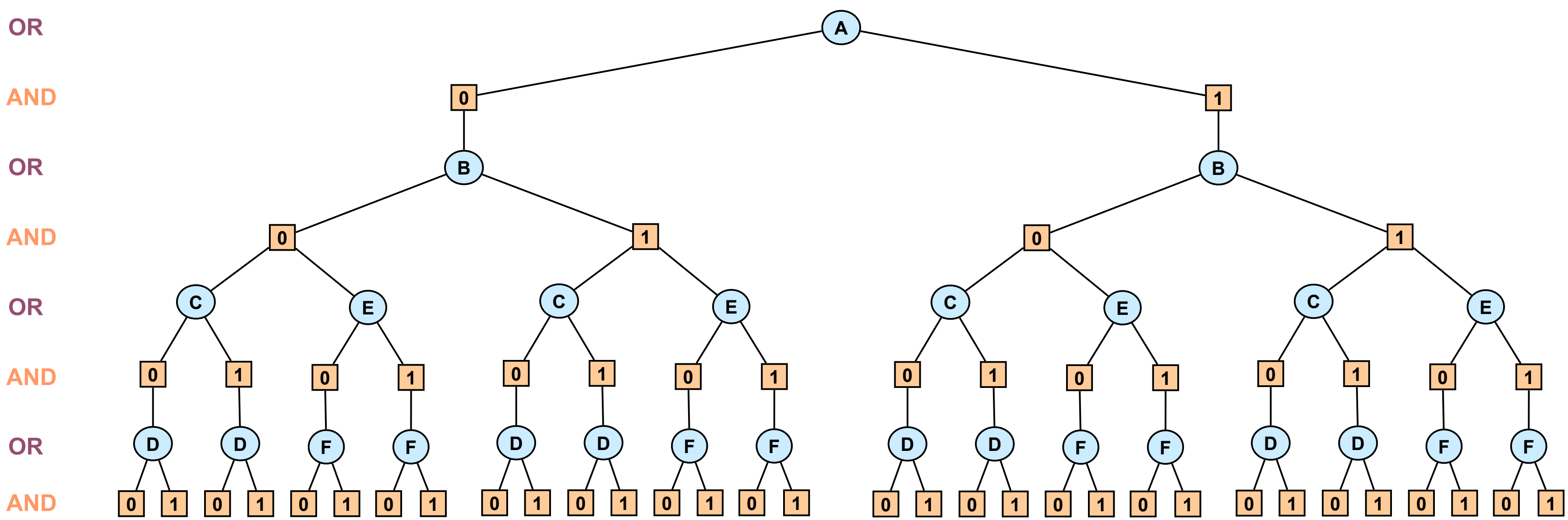
A	B	C	R_{ABC}
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

B	C	D	R_{BCD}
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**1**

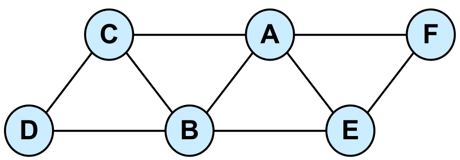
A	B	E	R_{ABE}
0	0	0	**1**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

A	E	F	R_{AEF}
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**





#CSP – AND/OR search tree

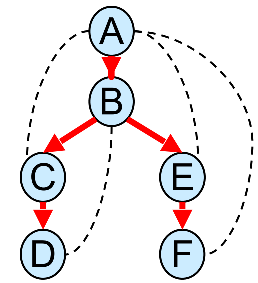


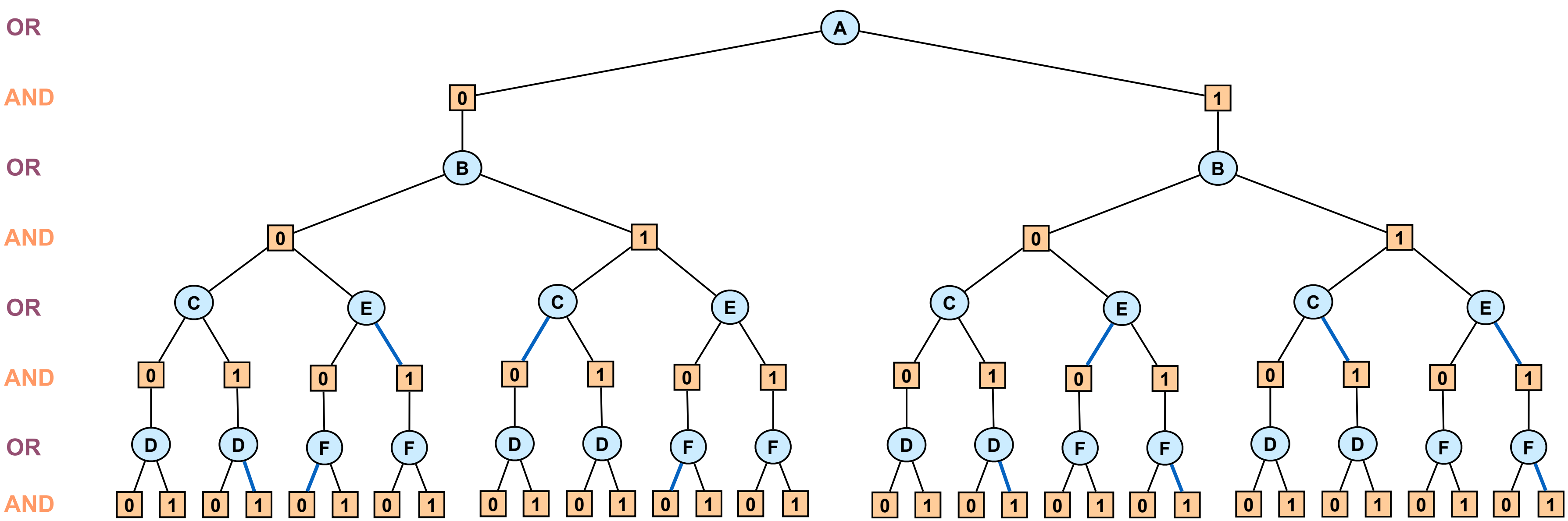
A	B	C	R_{ABC}
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

B	C	D	R_{BCD}
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**1**

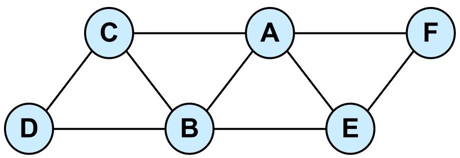
A	B	E	R_{ABE}
0	0	0	**1**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

A	E	F	R_{AEF}
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**





#CSP – AND/OR Tree DFS

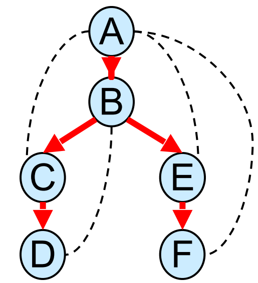


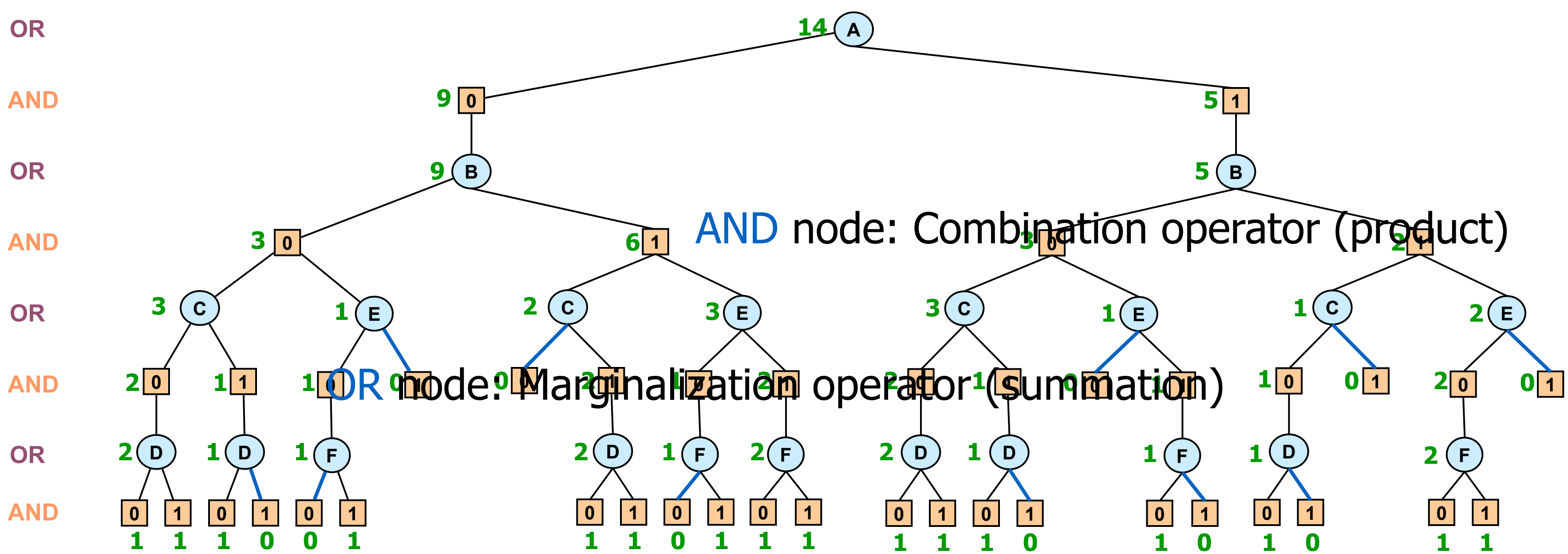
A	B	C	R_{ABC}
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

B	C	D	R_{BCD}
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**1**

A	B	E	R_{ABE}
0	0	0	**1**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

A	E	F	R_{AEF}
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

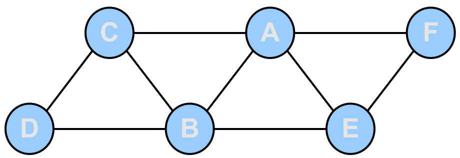


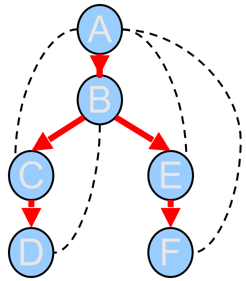


AND node: Combination operator (product)

OR node: Marginalization operator (summation)

AND/OR Tree search for COP





A	B	f_1
0	0	2
0	1	0
1	0	1
1	1	4

A	C	f_2
0	0	3
0	1	0
1	0	0
1	1	1

A	E	f_3
0	0	0
0	1	3
1	0	2
1	1	0

A	F	f_4
0	0	2
0	1	0
1	0	0
1	1	2

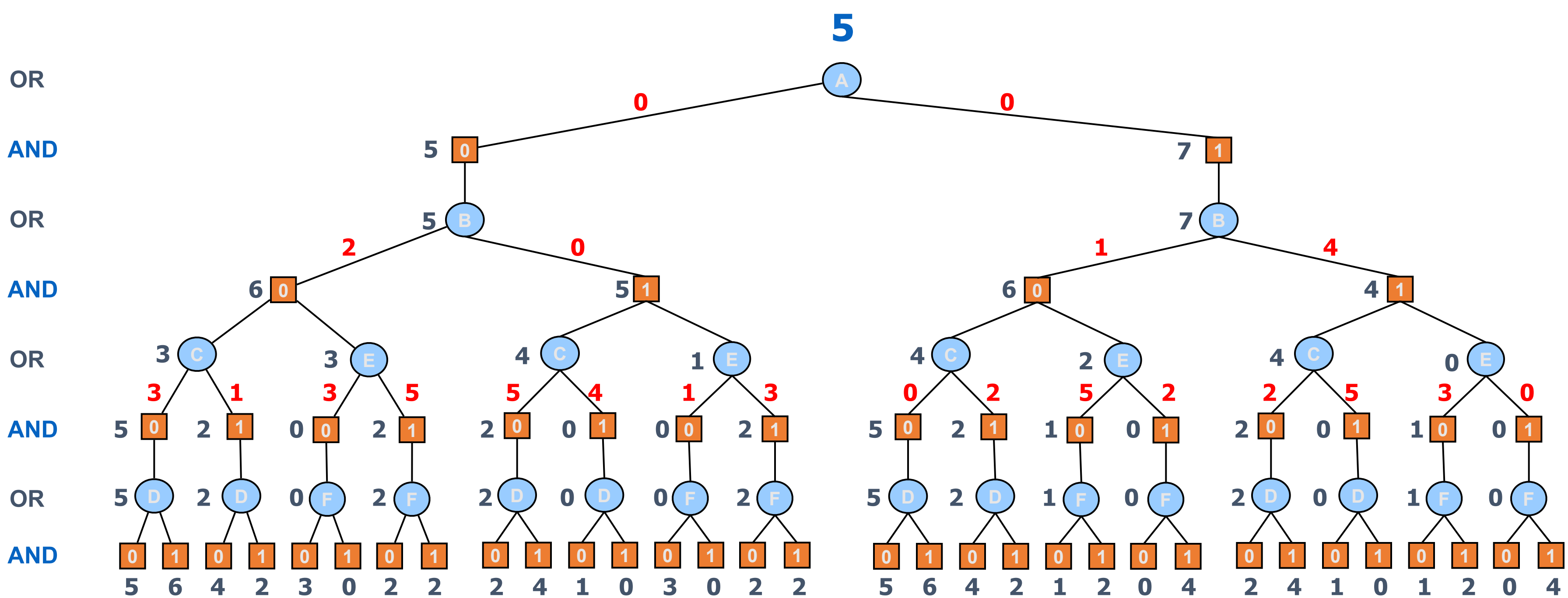
B	C	f_5
0	0	0
0	1	1
1	0	2
1	1	4

B	D	f_6
0	0	4
0	1	2
1	0	1
1	1	0

B	E	f_7
0	0	3
0	1	2
1	0	1
1	1	0

C	D	f_8
0	0	1
0	1	4
1	0	0
1	1	0

E	F	f_9
0	0	1
0	1	0
1	0	0
1	1	2



OR node = Marginalization operator (minimization)

AND node = Combination operator (summation)

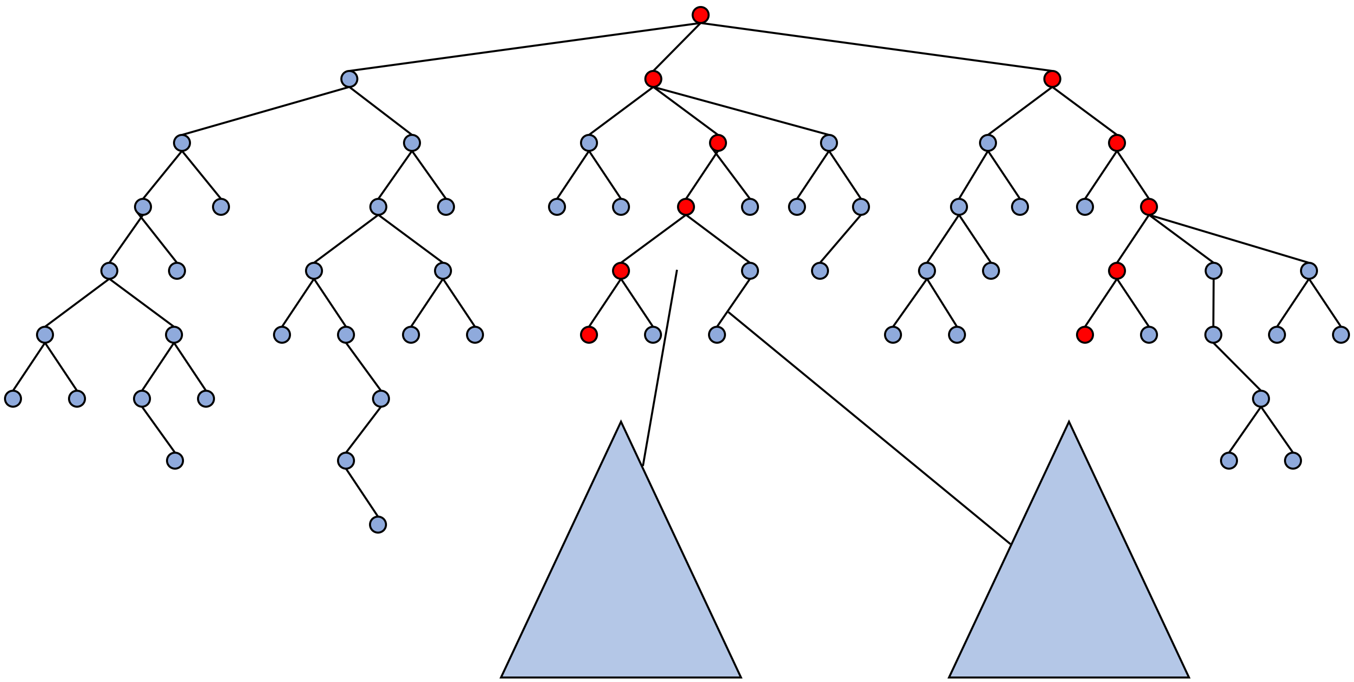
Summary of AND/OR search trees

- Based on a backbone pseudo-tree
- A solution is a subtree
- Each node has a **value** – cost of the optimal solution to the subproblem (computed recursively based on the values of the descendants)
- Solving a task = finding the value of the root node
- AND/OR search tree and algorithms are

  ([Freuder & Quinn85], [Collin, Dechter & Katz91], [Bayardo & Miranker95])
  - Space: O(n)
  - Time: O(exp(m)), where m is the depth of the pseudo-tree
  - Time: O(exp(w* log n))
  - BFS is time and space O(exp(w* log n)

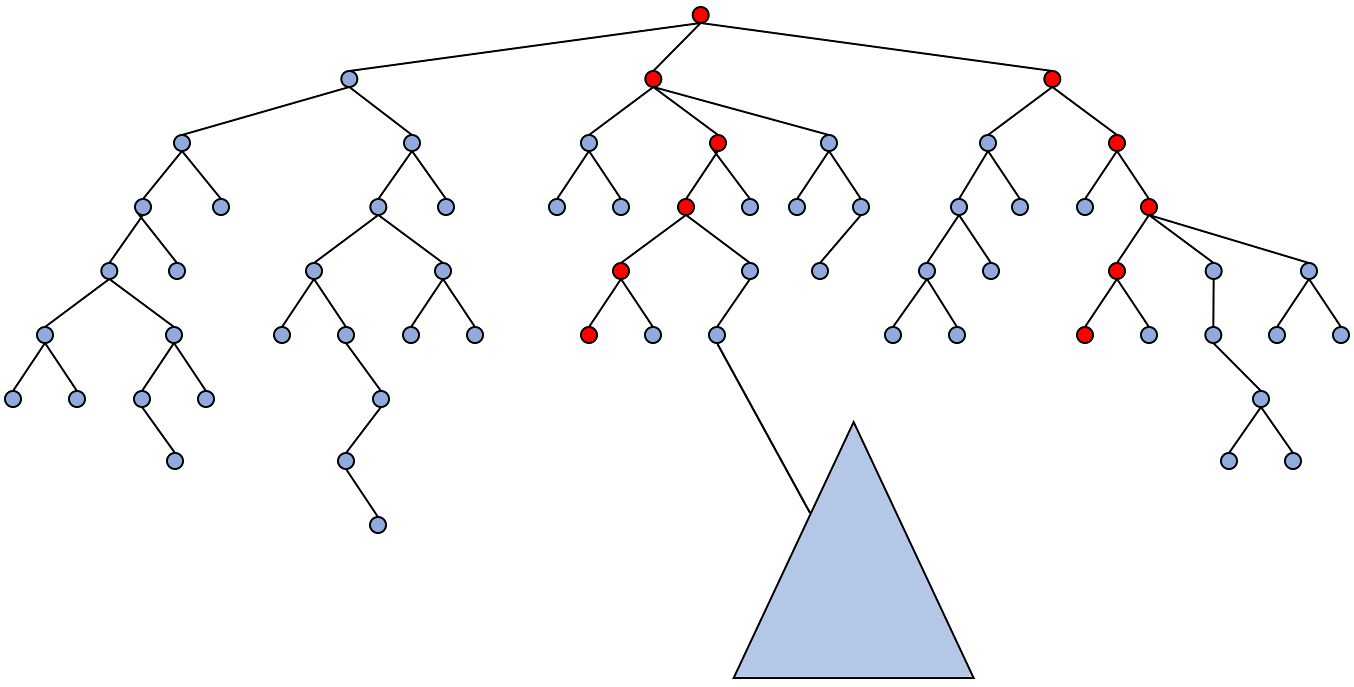
From search trees to search graphs

- Any two nodes that root identical subtrees or subgraphs can be merged

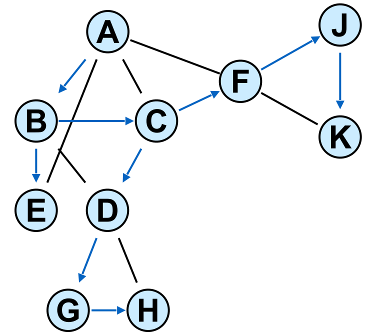


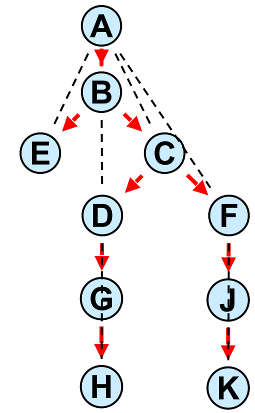
From search trees to search graphs

- Any two nodes that root identical subtrees or subgraphs can be merged

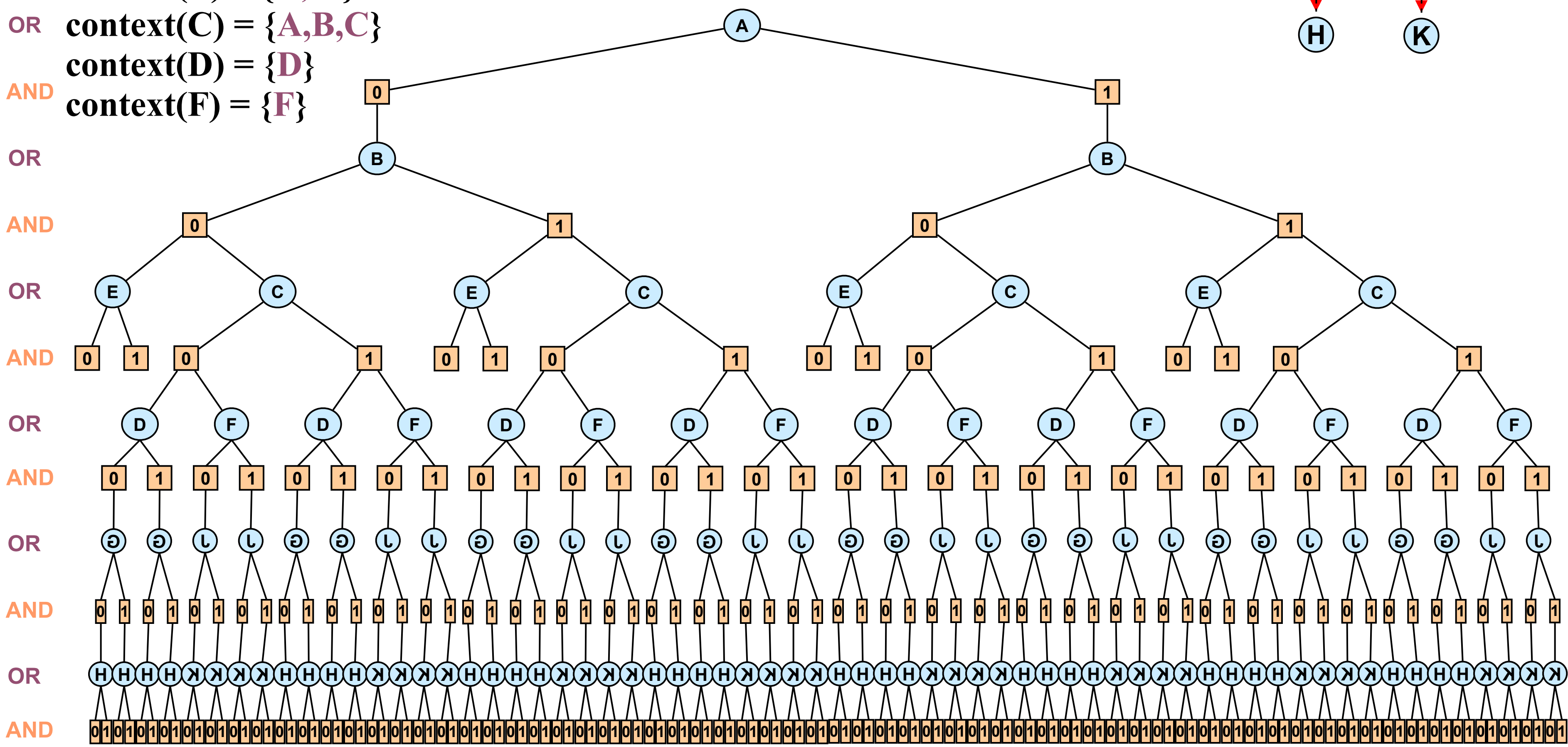


Caching

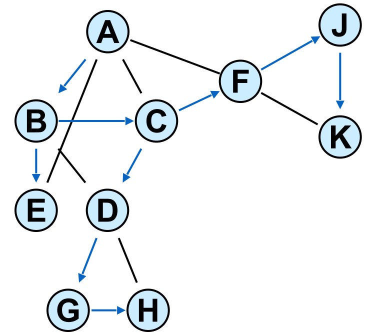


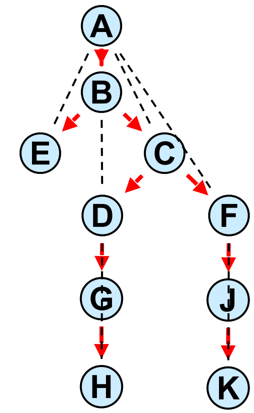


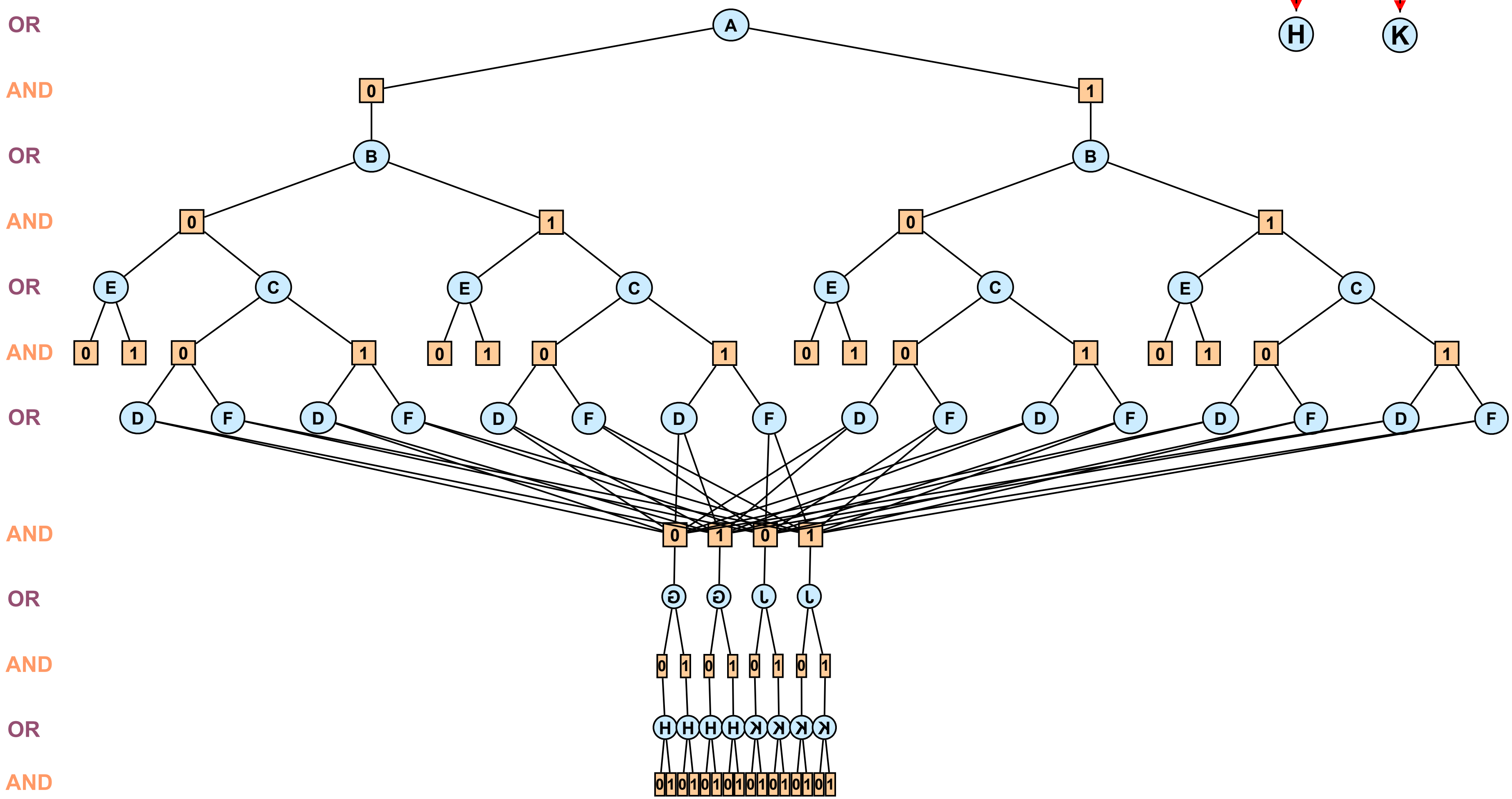
context(B) = {A, B}
context(C) = {A,B,C}
context(D) = {D}
context(F) = {F}



An AND/OR graph: Caching goods



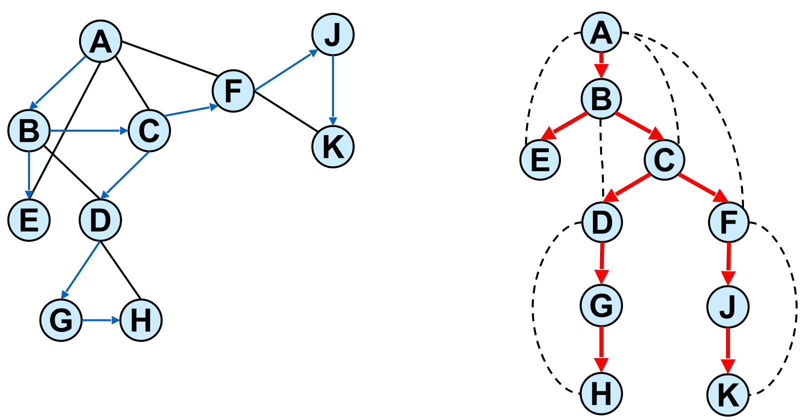




Context-based Caching

- Caching is possible when context is the same

- context = parent-separator set in induced pseudo-graph
  = current variable + parents connected to subtree below



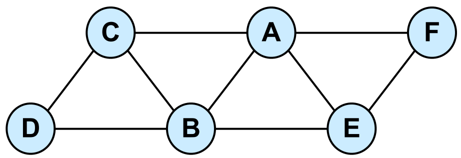
context(B) = {A, B}

context(c) = {A,B,C}

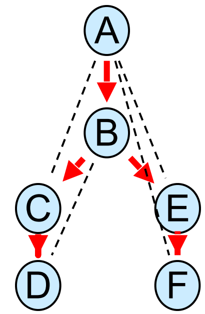
context(D) = {D}

context(F) = {F}

AND/OR Search Graph

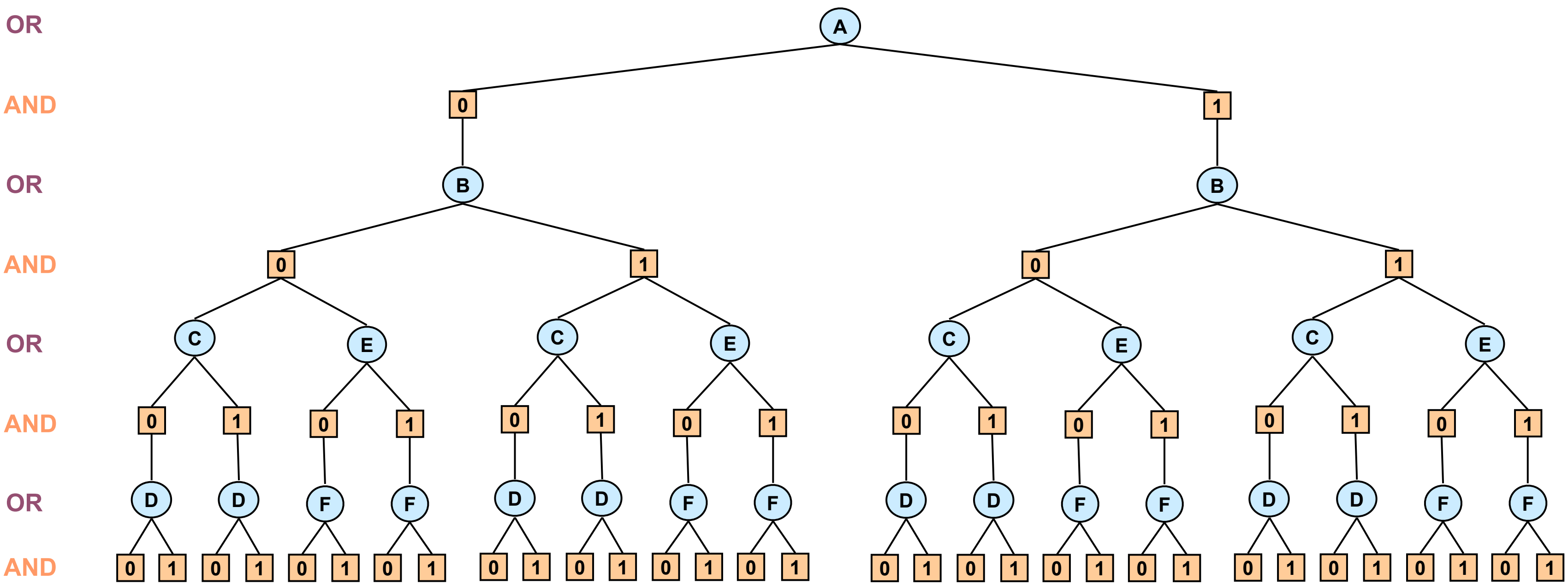


Primal graph

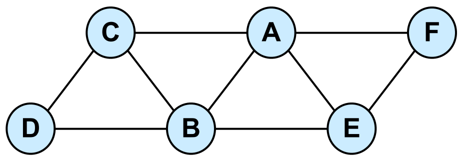


Pseudo-tree

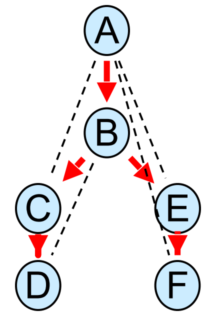
context(A) = {**A**}
context(B) = {**B,A**}
context(C) = {**C,B**}
context(D) = {**D**}
context(E) = {**E,A**}
context(F) = {**F**}



AND/OR Search Graph

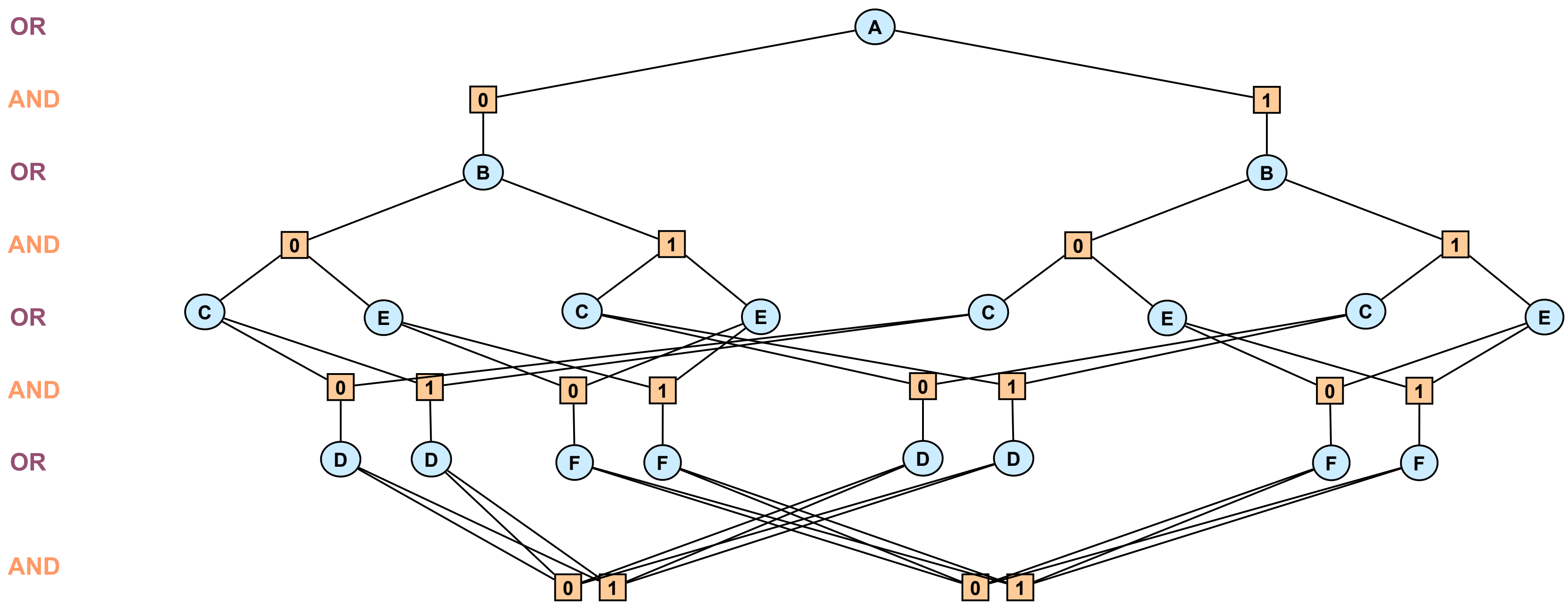


Primal graph

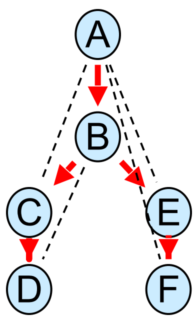


Pseudo-tree

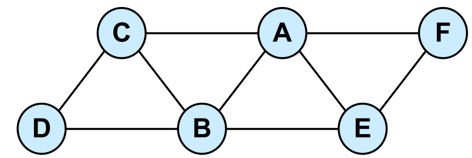
context(A) = {**A**}
context(B) = {**B,A**}
context(C) = {**C,B**}
context(D) = {**D**}
context(E) = {**E,A**}
context(F) = {**F**}



Context-based caching



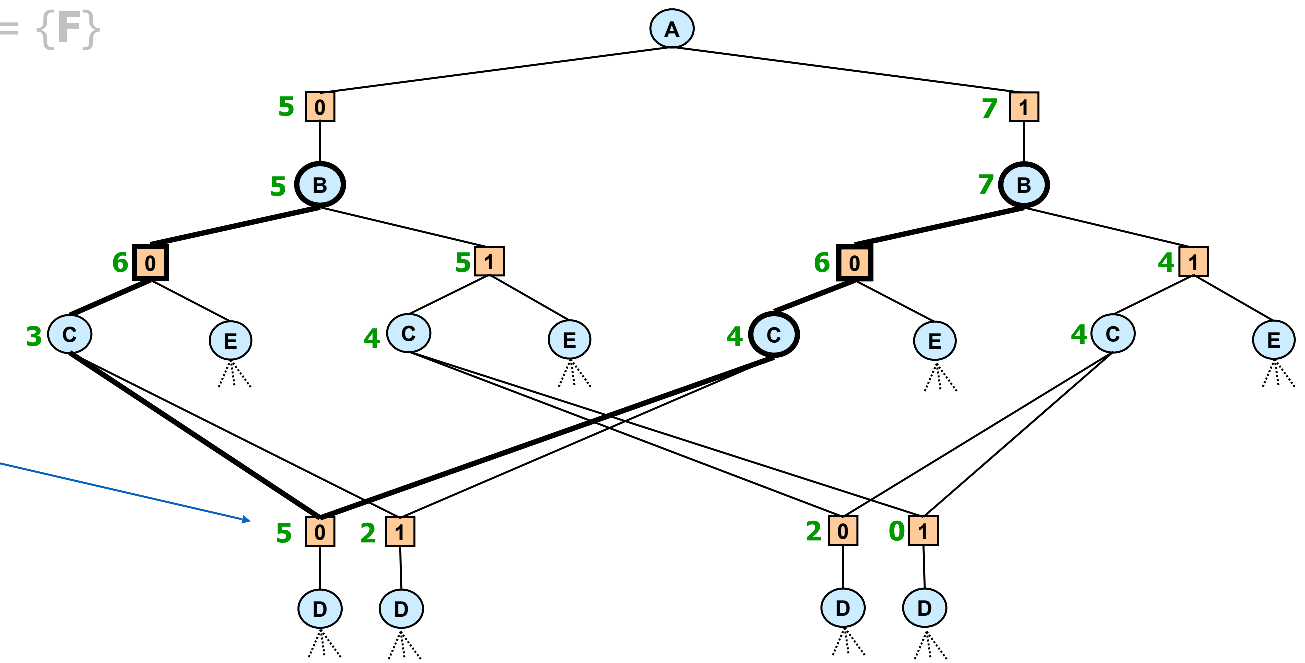
context(A) = {**A**}
context(B) = {**B,A**}
context(C) = {C,B}
context(D) = {**D**}
context(E) = {**E,A**}
context(F) = {**F**}



Primal graph

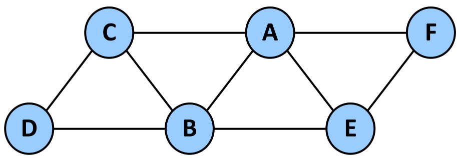
Cache Table (C)

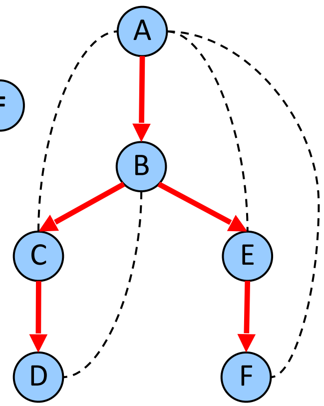
B	C	Value
0	0	5
0	1	2
1	0	2
1	1	0



Space: **O(exp(2))**

AND/OR Search Graph (Optimization)





A	B	f_1
0	0	2
0	1	0
1	0	1
1	1	4

A	C	f_2
0	0	3
0	1	0
1	0	0
1	1	1

A	E	f_3
0	0	0
0	1	3
1	0	2
1	1	0

A	F	f_4
0	0	2
0	1	0
1	0	0
1	1	2

B	C	f_5
0	0	0
0	1	1
1	0	2
1	1	4

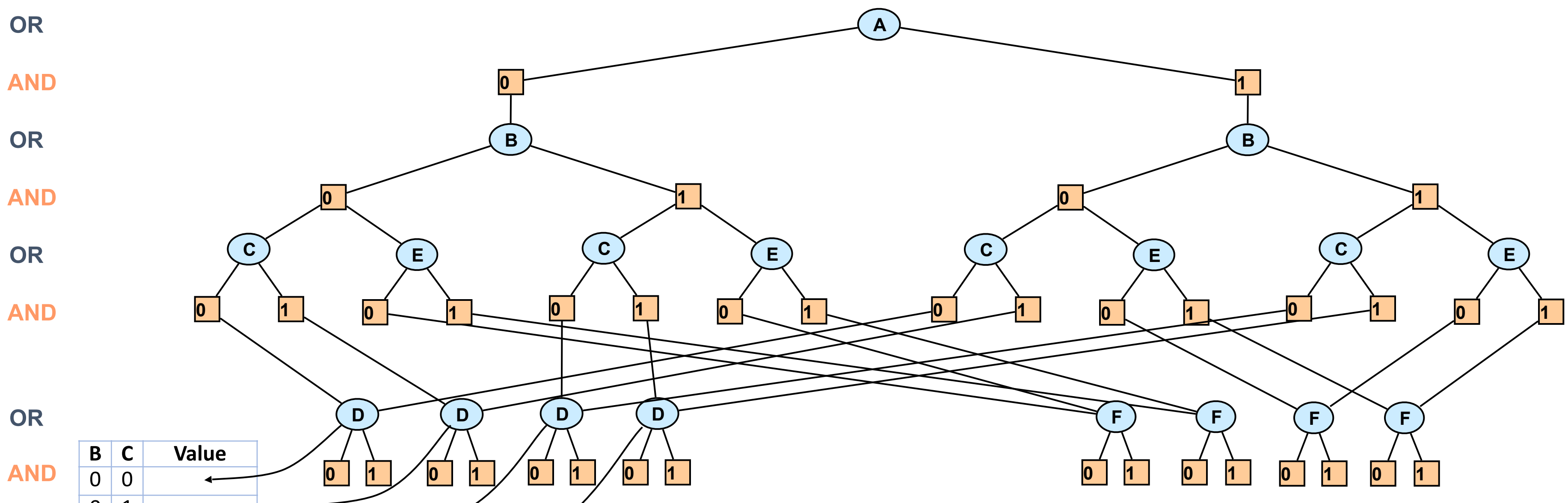
B	D	f_6
0	0	4
0	1	2
1	0	1
1	1	0

B	E	f_7
0	0	3
0	1	2
1	0	1
1	1	0

C	D	f_8
0	0	1
0	1	4
1	0	0
1	1	0

E	F	f_9
0	0	1
0	1	0
1	0	0
1	1	2

Objective function: $$F^* = \min_x \sum_\alpha f_\alpha(x_\alpha)$$



B	C	Value
0	0	
0	1	
1	0	
1	1	

Cache table for **D**

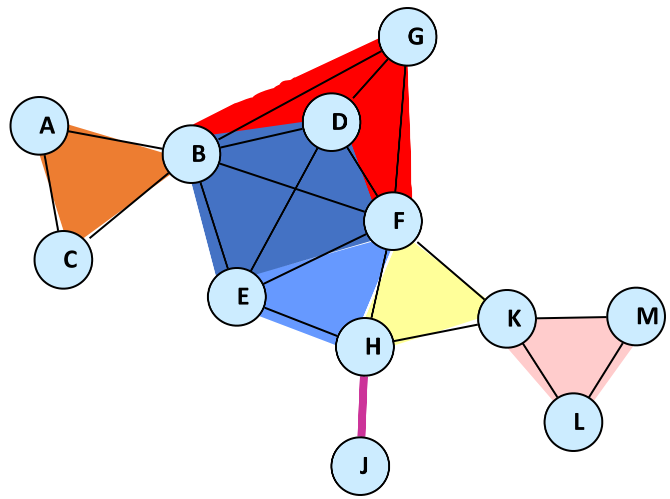
Context minimal AND/OR search graph

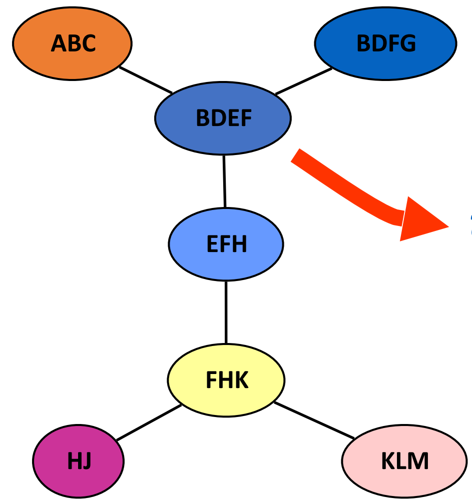
Complexity of AND/OR Graph

- **Theorem:** Traversing the AND/OR search graph is time and space exponential in the induced width/ tree-width.

- If applied to the OR graph complexity is time and space exponential in the path-width.

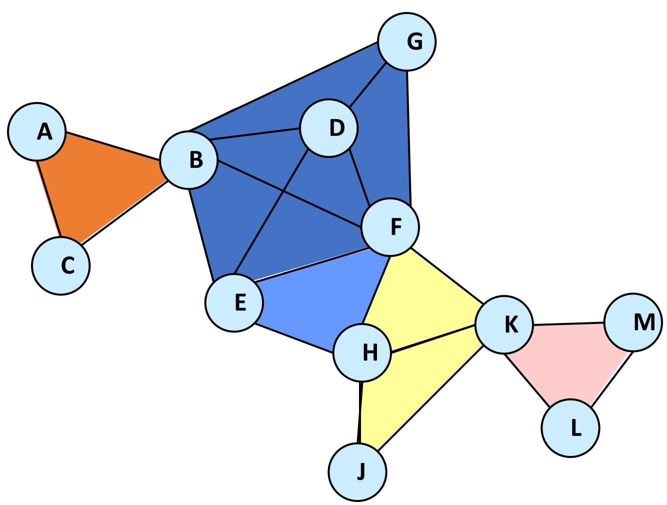
Treewidth vs. Pathwidth

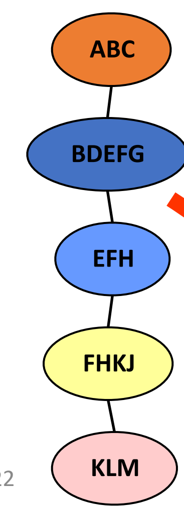




***treewidth* = 3**

= (max cluster size) - 1

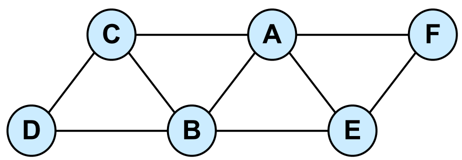




***pathwidth* = 4**

= (max cluster size) - 1

#CSP – AND/OR search tree

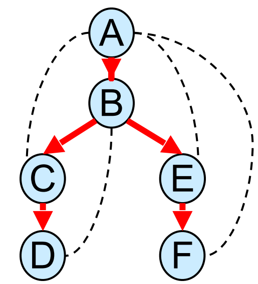


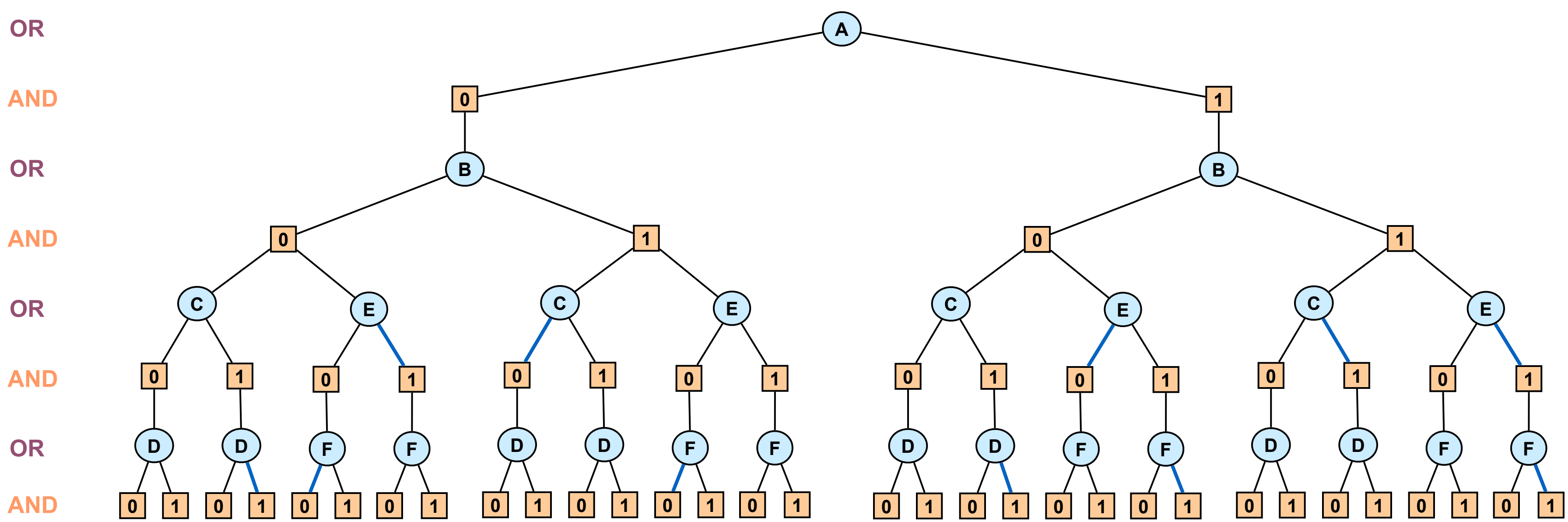
A	B	C	R_{ABC}
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

B	C	D	R_{BCD}
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**1**

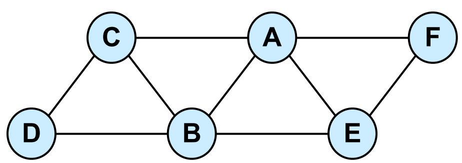
A	B	E	R_{ABE}
0	0	0	**1**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

A	E	F	R_{AEF}
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**





#CSP – AND/OR tree dfs

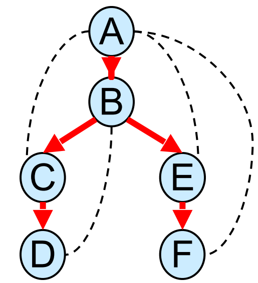


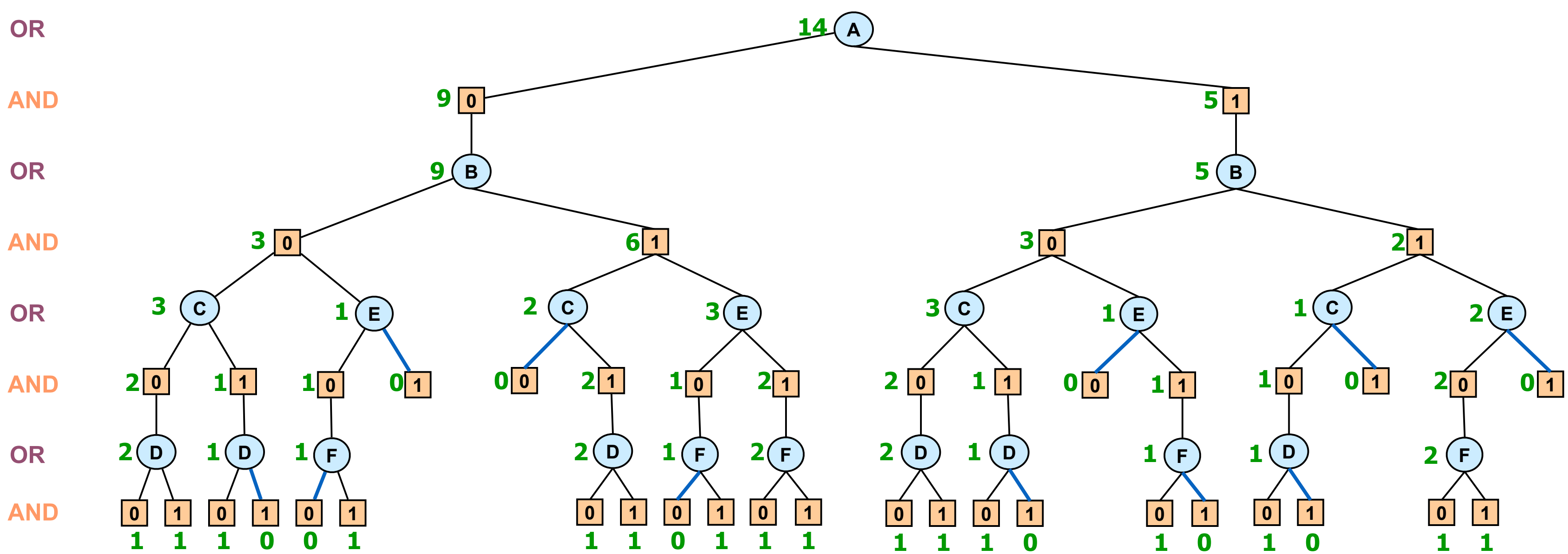
A	B	C	R_{ABC}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

B	C	D	R_{BCD}
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

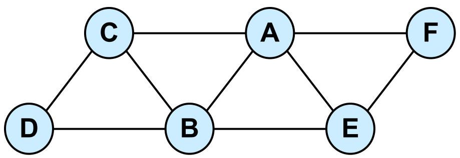
A	B	E	R_{ABE}
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A	E	F	R_{AEF}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





#CSP – AND/OR search graph (Caching goods)

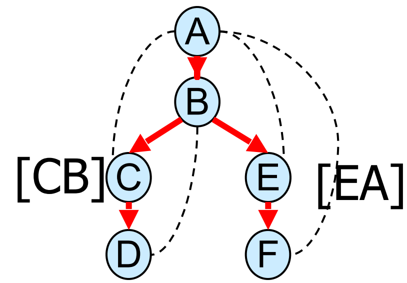


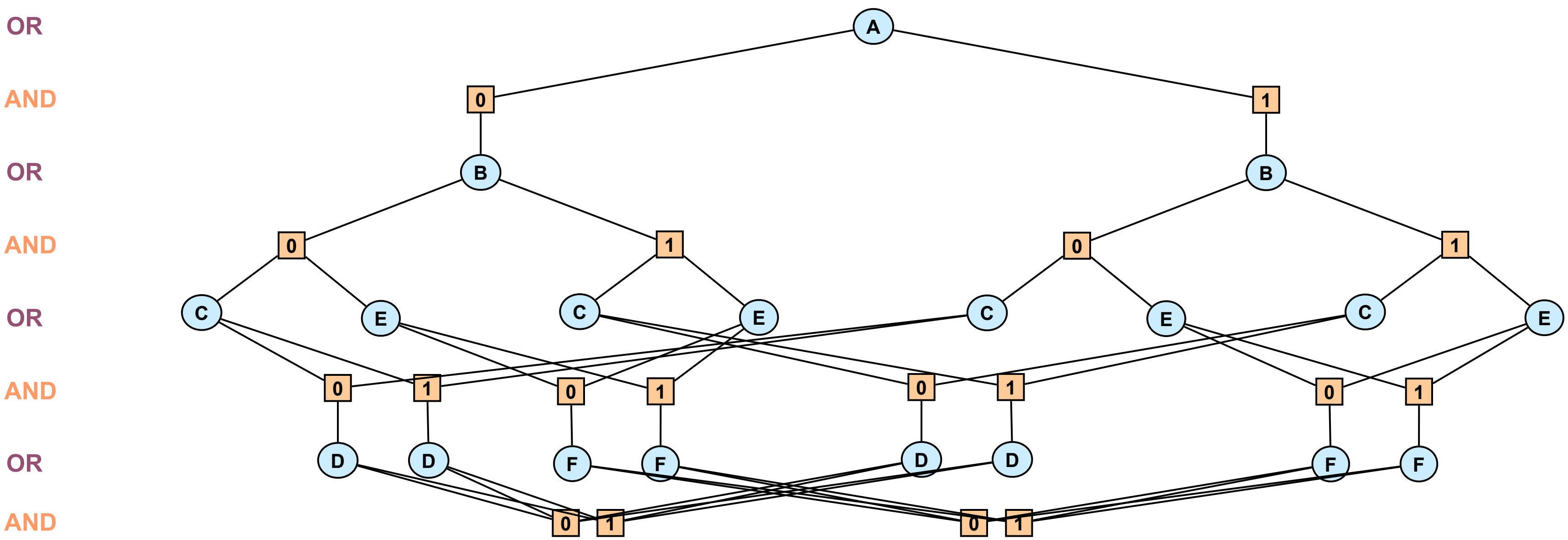
A	B	C	R_{ABC}
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

B	C	D	R_{BCD}
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**1**

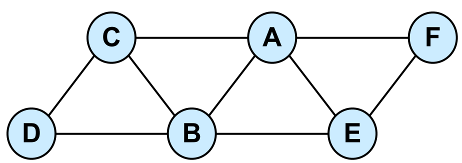
A	B	E	R_{ABE}
0	0	0	**1**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

A	E	F	R_{AEF}
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**





#CSP – AND/OR search graph (Caching goods)

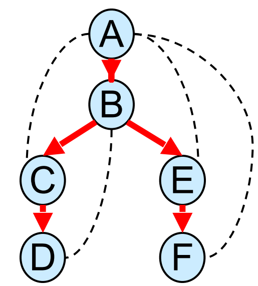


A	B	C	R_{ABC}
0	0	0	**1**
0	0	1	**1**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

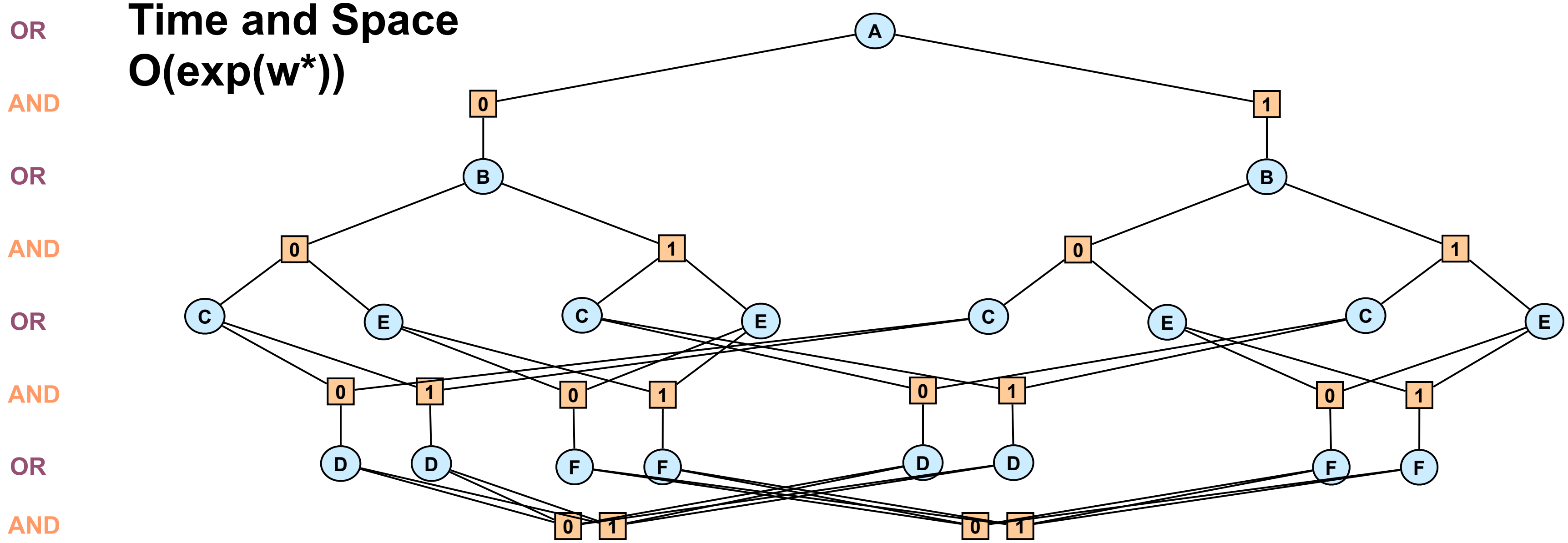
B	C	D	R_{BCD}
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**1**

A	B	E	R_{ABE}
0	0	0	**1**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

A	E	F	R_{AEF}
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

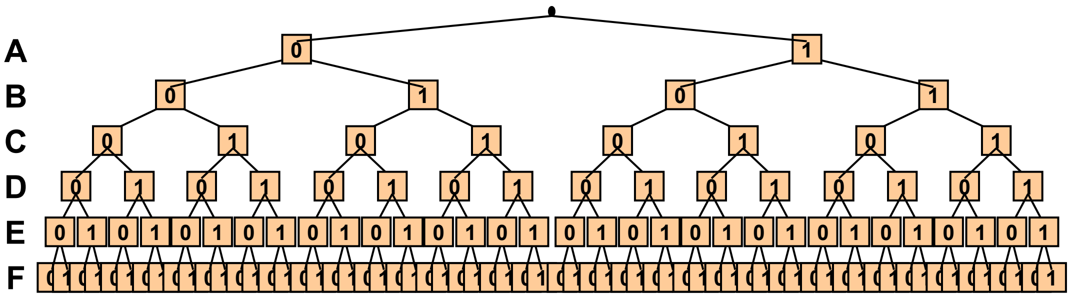


Time and Space O(exp(w*))



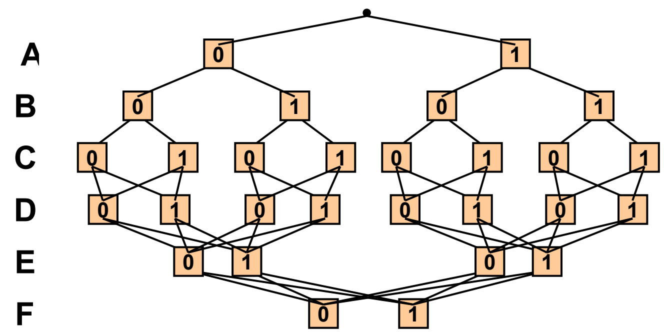
Space O(exp(sep-w*))

All Four Search Spaces



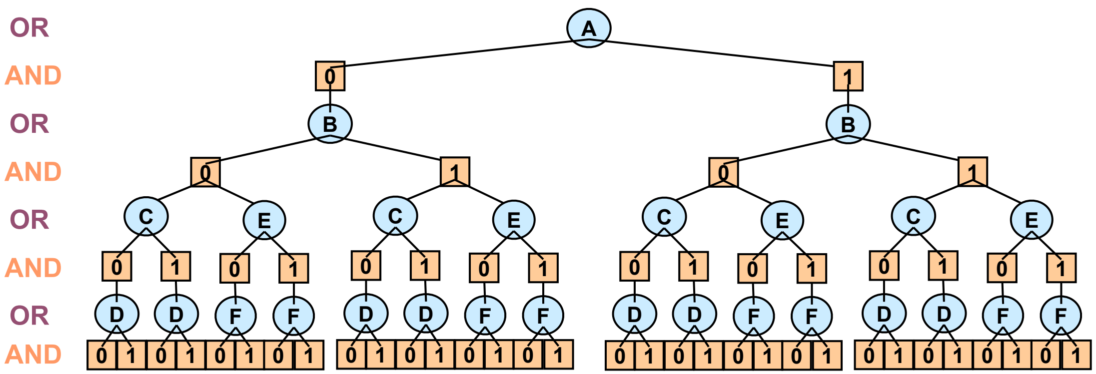
Full OR search tree

126 nodes



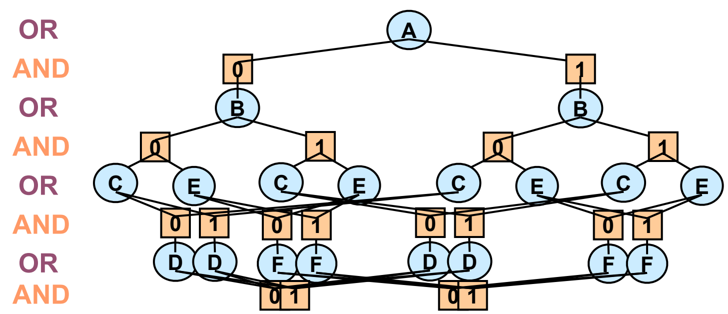
Context minimal OR search graph

28 nodes



Full AND/OR search tree

54 AND nodes



Context minimal AND/OR search graph

18 AND nodes

AND/OR vs. OR dfs algorithms

k = domain size
m = pseudo-tree depth
n = number of variables
w^*= induced width
pw^*= path width

- AND/OR tree
  - Space: $O(n)$
  - Time: $O(n\ k^m)$
    $O(n\ k^{w^* \log n})$
    (Freuder85; Bayardo95; Darwiche01)

- OR tree
  - Space: $O(n)$
  - Time: $O(k^n)$

- AND/OR graph
  - Space: $O(n\ k^{w^*})$
  - Time: $O(n\ k^{w^*})$

- OR graph
  - Space: $O(n\ k^{pw^*})$
  - Time: $O(n\ k^{pw^*})$

Searching AND/OR graphs

- AO(i): searches depth-first, cache i-context
  - i = the max size of a cache table (i.e. number of variables in a context)

i=0 → i=w*

Space: O(n)

Time: O(exp(w* log n))

Space: O(exp w*)

Time: O(exp w*)

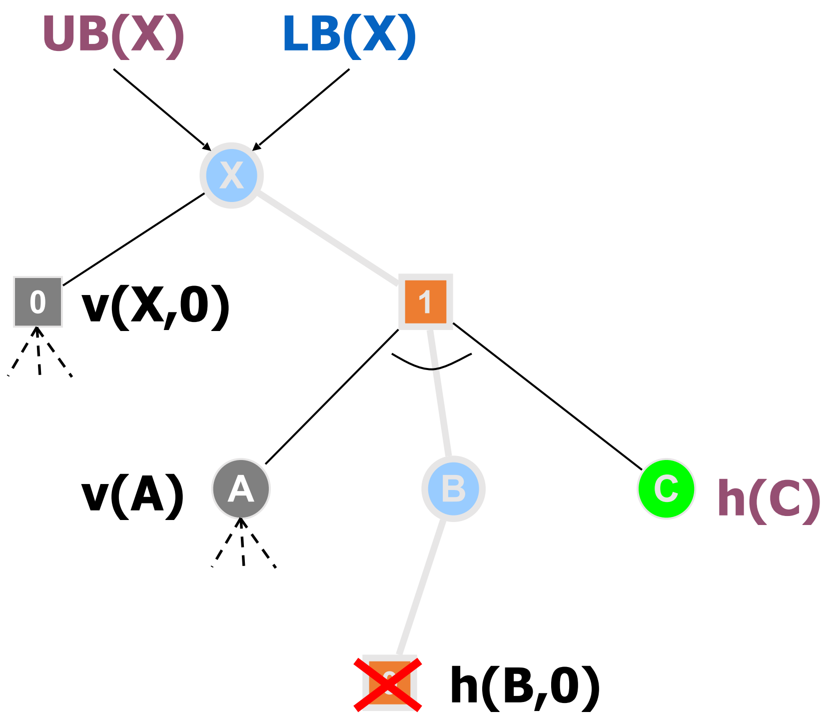
AO(i) time complexity?

AND/OR branch-and-bound (AOBB)

- Associate each node **n** with a static heuristic estimate h(n) of v(n)
  - h(n) is a lower bound on the value v(n)

- For every node **n** in the search tree:
  - ub(n) – current best solution cost rooted at n
  - lb(n) – lower bound on the minimal cost at n

Lower/Upper bounds



UB(X) = best cost below X (i.e. v(X,0))

LB(X) = LB(X,1)

LB(X,1) = l(X,1) + v(A) + h(C) + LB(B)

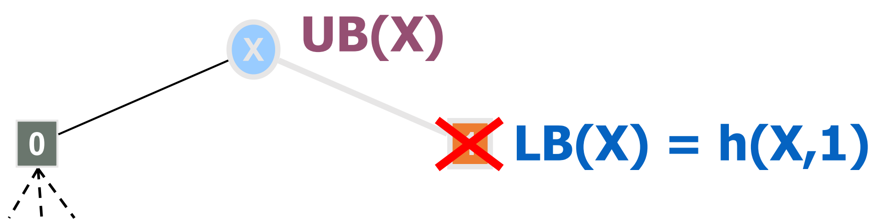
LB(B) = LB(B,0)

LB(B,0) = h(B,0)

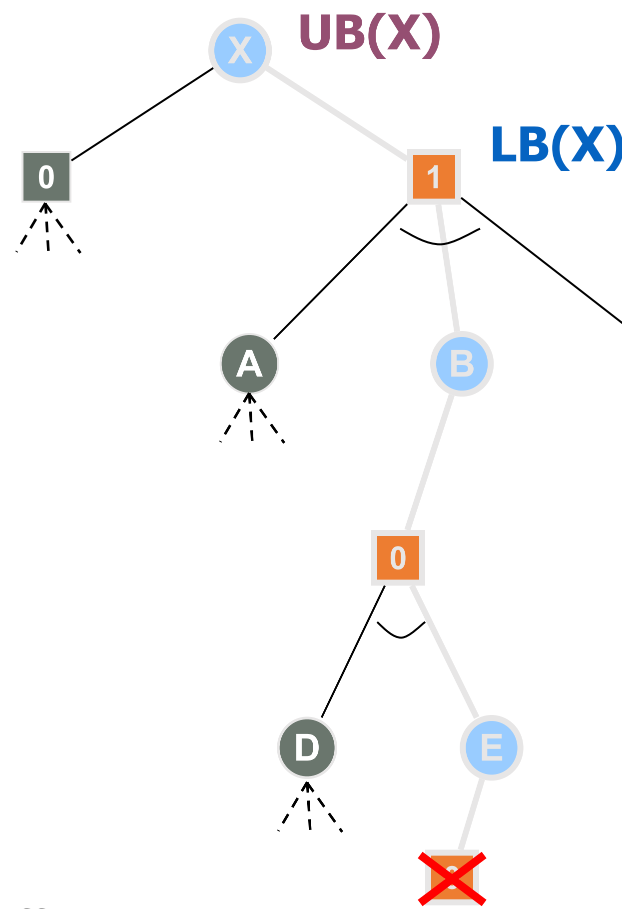
Prune below AND node (B,0) if LB(X) ≥ UB(X)

Shallow/deep cutoffs

Prune if LB(X) ≥ UB(X)



Shallow cutoff



Reminiscent of **Minimax** shallow/deep cutoffs

Deep cutoff

Summary of AOBB

- Traverses the AND/OR search tree in a depth-first manner
- Lower bounds computed based on heuristic estimates of nodes at the frontier of search, as well as the values of nodes already explored
- Prunes the search space as soon as an upper-lower bound violation occurs

Heuristics for AND/OR

- In the AND/OR search space **h(n)** can be computed using any heuristic. We used:
  - Static Mini-Bucket heuristics
  - Dynamic Mini-Bucket heuristics
  - Maintaining FDAC [Larrosa & Schiex03]
    (full directional soft arc-consistency)

Empirical evaluation

- Tasks
  - Solving WCSPs
  - Finding the MPE in belief networks

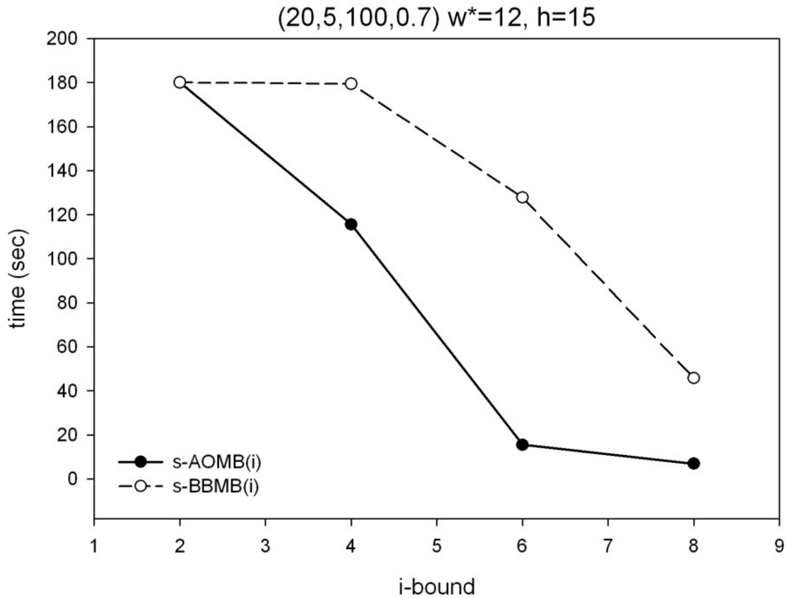
- Benchmarks (WCSP)
  - Random binary WCSPs
  - RLFAP networks (CELAR6)
  - Bayesian Networks Repository

- Algorithms
  - s-AOMB(i), d-AOMB(i), AOMFDAC
  - s-BBMB(i), d-BBMB(i), BBMFDAC
  - Static variable ordering (dfs traversal of the pseudo-tree)

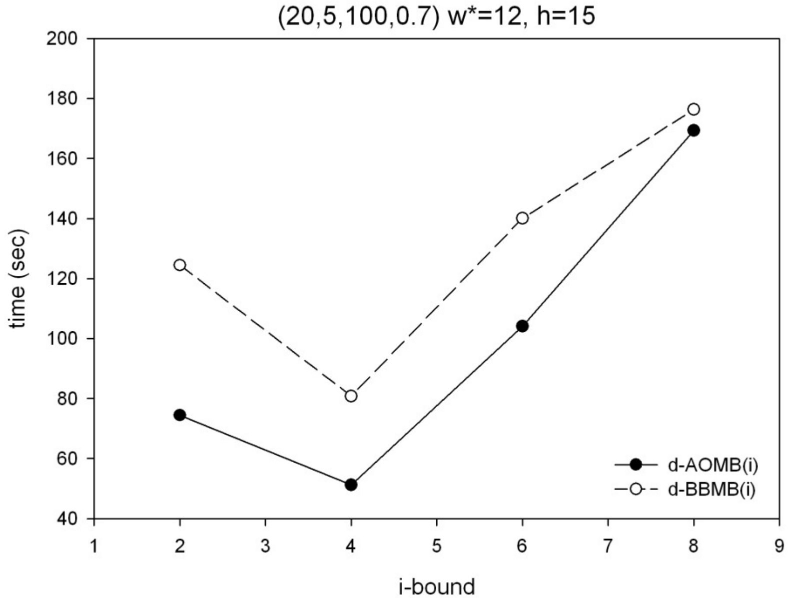
Random binary wcsps

(Marinescu and Dechter, 2005)

S-AOMB vs S-BBMB



D-AOMB vs D-BBMB

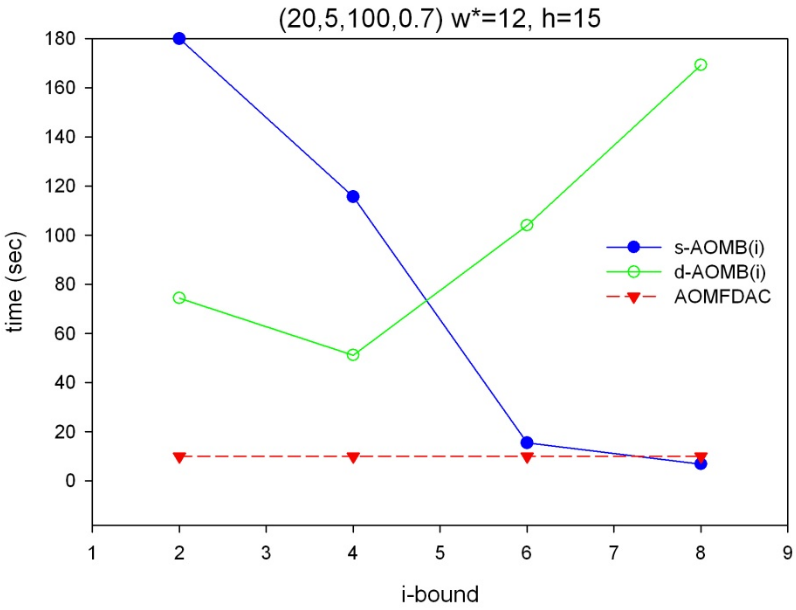


Random networks with n=20 (number of variables), d=5 (domain size), c=100 (number of constraints), t=70% (tightness). Time limit 180 seconds.

AO search is superior to OR search

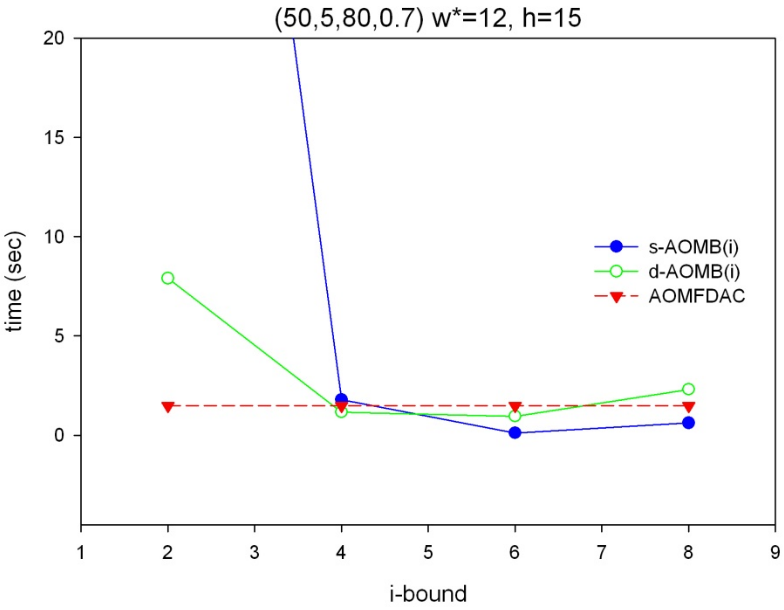
Random binary wcsps (contd.)

dense



n=20 (variables), d=5 (domain size), c=100 (constraints), t=70% (tightness)

sparse



n=50 (variables), d=5 (domain size), c=80 (constraints), t=70% (tightness)

AOMB for large i is competitive with AOMFDAC

Resource allocation

Radio Link Frequency Assignment Problem (RLFAP)

Instance	BBMFDAC		AOMFDAC	
	time (sec)	nodes	time (sec)	nodes
CELAR6-SUB0	2.78	1,871	**1.98**	435
CELAR6-SUB1	2,420.93	364,986	**981.98**	180,784
CELAR6-SUB2	8,801.12	19,544,182	**1,138.87**	175,377
CELAR6-SUB3	38,889.20	91,168,896	**4,028.59**	846,986
CELAR6-SUB4	84,478.40	6,955,039	**47,115.40**	4,643,229

CELAR6 sub-instances

AOMFDAC is superior to **OR**MFDAC

Bayesian networks repository

Network (n,d,w*,h)	Algorithm	i=2 time	i=2 nodes	i=3 time	i=3 nodes	i=4 time	i=4 nodes	i=5 time	i=5 nodes
Barley (48,67,7,17)	**s-AOMB(i)**	-	8.5M	-	7.6M	46.22	807K	**0.563**	9.6K
	s-BBMB(i)	-	16M	-	18M	-	17M	-	14M
	d-AOMB(i)	-	79K	136.0	23K	12.55	667	45.95	567
	d-BBMB(i)	-	2.2M	-	1M	346.1	76K	-	86K
Munin1 (189,21,11,24)	**s-AOMB(i)**	57.36	1.2M	12.08	260K	7.203	172K	**1.657**	43K
	s-BBMB(i)	-	8.5M	-	9M	-	10M	-	8M
	d-AOMB(i)	66.56	185K	12.47	8.1K	10.30	1.6K	11.99	523
	d-BBMB(i)	-	405K	-	430K	-	235K	14.63	917
Munin3 (1044,21,7,25)	**s-AOMB(i)**	-	5.9M	-	4.9M	1.313	17K	**0.453**	6K
	s-BBMB(i)	-	1.4M	-	1.2M	-	316K	-	1.5M
	d-AOMB(i)	-	2.3M	68.64	58K	3.594	5.9K	2.844	3.8K
	d-BBMB(i)	-	33K	-	125K	-	52K	-	31K

Time limit 600 seconds

available at http://www.cs.huji.ac.il/labs/compbio/Repository

Static AO is better with accurate heuristic (large i)

Outline

- **Introduction**
  - Optimization tasks for graphical models
  - Solving optimization problems with inference and search
- **Inference**
  - Bucket elimination, dynamic programming
  - Mini-bucket elimination
- **Search**
  - Branch and bound and best-first
  - Lower-bounding heuristics
  - AND/OR search spaces